RNN SEISMIC VELOCITY MODEL BUILDING: IMPROVING GENERALIZATION USING A
FREQUENCY-STEPPING APPROACH AND HYBRID TRAINING DATA

by

Hani Ataiq Alzahrani
A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Geophysics).

Golden, Colorado
Date ______________________

Signed: ______________________

Hani Ataiq Alzahrani

Signed: ______________________

Dr. Jeffrey Shragge
Thesis Advisor

Golden, Colorado
Date ______________________

Signed: ______________________

Dr. Paul Sava
Department Head
Department of Geophysics
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 lead to the possibility of estimating high-resolution subsurface velocity models at a low computational cost. However, a significant challenge of ANNs is training generalization, which is the ability of an ANN to apply the learning from the training process to evaluate test data not previously encountered during the training process. In the context of velocity model building, this means learning the relationship between velocity models and the corresponding seismic data from a set of training data, and then using acquired seismic data to accurately estimate unknown velocity models. While generalizing to testing models with structures similar to those found in the training data has become a manageable task as evidenced in the recent literature, extending generalization to more realistic scenarios where testing models may exhibit drastically different velocity structures and/or distributions than those in the training data set remains an important and ongoing research challenge. To address this issue, this thesis develops and present the applications of a multi-scale approach inspired by physics-based full-waveform inversion that uses recurrent neural networks to invert frequency-domain seismic data using a frequency-stepping scheme. The input data consist of a sequence of seismic frequency slices that are fed to the network progressively from the lowest available to the highest usable in the data. I combine this approach with a hybrid training approach that merges background velocity gradient models with purely geometrical and geologically realistic model structures. This combination increases the range of spatial wavenumbers as well as the variability of geological structures present in the training data. I demonstrate the potential for improved generalization by comparing the model estimates results from two trained networks: one using a hybrid set of models, the other with only geological models. I test the two networks using subsets of the community BP2004 benchmark model with complex salt structures fully absent from the models used in the training process. Qualitative analysis shows that models recovered using hybrid training data are significantly more accurate than those recovered using geological training data alone, with arbitrarily shaped salt bodies being accurately delineated by the trained hybrid network. In addition, I demonstrate through a quantitative SSIM metric analysis that the developed RNN extends the range of structures recoverable by the trained ANN. The developed approach illustrates the potential of neural networks to learn the seismic velocity model building problem at a general level from a representative set of training models, and opens the way for more research into improving the design of non-geological training data to further improve network generalization.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td>xiii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>xiv</td>
</tr>
<tr>
<td>CHAPTER 1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Enter Machine Learning</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Research Challenge</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Thesis Outline</td>
<td>5</td>
</tr>
<tr>
<td>CHAPTER 2 SEISMIC VELOCITY MODEL BUILDING USING NEURAL NETWORKS:</td>
<td>8</td>
</tr>
<tr>
<td>TRAINING DATA DESIGN AND LEARNING Generalization</td>
<td></td>
</tr>
<tr>
<td>2.1 Abstract</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Introduction</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Generating Training Data</td>
<td>10</td>
</tr>
<tr>
<td>2.3.1 Semi-Geological Models - Types 1a/1b</td>
<td>10</td>
</tr>
<tr>
<td>2.3.2 Squares-Only Velocity Models - Types 2a/2b</td>
<td>11</td>
</tr>
<tr>
<td>2.3.3 Data Set Characteristics</td>
<td>11</td>
</tr>
<tr>
<td>2.4 Artificial Neural Networks</td>
<td>14</td>
</tr>
<tr>
<td>2.4.1 Learning Generalization</td>
<td>16</td>
</tr>
<tr>
<td>2.4.2 Convolutional Neural Networks</td>
<td>16</td>
</tr>
<tr>
<td>2.5 Experiments</td>
<td>17</td>
</tr>
<tr>
<td>2.5.1 ANN Architecture</td>
<td>17</td>
</tr>
<tr>
<td>2.5.2 Training</td>
<td>21</td>
</tr>
<tr>
<td>2.5.3 Experiment 1: CNN Trained on Type 1a/1b Velocity Models</td>
<td>22</td>
</tr>
<tr>
<td>2.5.4 Experiment 2: CNN Trained on Type 2a/2b Velocity Models</td>
<td>24</td>
</tr>
<tr>
<td>2.5.5 Scale Analysis of Type 2a/2b models</td>
<td>29</td>
</tr>
</tbody>
</table>
Figure 2.1 A number of members from the two sets of randomly generated “semi-geological” velocity models, along with corresponding “pseudo-well” cross-sections extracted at the dashed line of each model. Examples of (a) Type 1a velocity models with no velocity gradients and (b) Type 1b velocity models where some randomly selected layers have velocity gradients.

Figure 2.2 A number of members from the two sets of randomly generated “semi-geological” velocity models, along with corresponding “pseudo-well” cross-sections extracted at the dashed line of each model. Examples of (a) Type 1a velocity models with no velocity gradients and (b) Type 1b velocity models where some randomly selected layers have velocity gradients.

Figure 2.3 A sketch of a simple feed-forward neural network (FFNN). (a) The connections between the input layer and the first neuron in the hidden layer demonstrate how the value of that neuron is computed. (b) All connections from the input layer to the output layer.

Figure 2.4 A sketch of an ANN that has (a) an input layer, (b)-(d) a hidden convolutional layer, (e)-(f) a max-pooling layer, and (g) an output layer. (c) Shows how the convolution process is carried out using a sliding convolution operator, with the output of each convolution step placed in the corresponding cell in (d). Similarly, (e) shows a sliding max-pooling filter, with the output of each step placed in the corresponding cell in (f). The max-pooling layer is in turn connected to the output layer (g).

Figure 2.5 A sketch showing the architecture of the ANN used in all the experiments in this work.

Figure 2.6 A sketch showing how seismic data for the $i$-th frequency ($0 \leq i < 30$) are sorted for input into a single channel of the convolutional layer. $n_x$ is the number of grid points in the $x$ direction, $N_s$ and $N_r$ denote the number of sources and receivers, respectively.

Figure 2.7 A sketch demonstrating how frequency-domain seismic data are injected into the input layer of the ANN. The convolutional layer has 30 channels, where each channel is assigned to a single frequency in the input seismic data.

Figure 2.8 Learning curves for the four CNNs trained using (a) Type 1a data, (b) Type 1b data, (c) Type 2a data, and (d) Type 2b data.

Figure 2.9 Testing results of the neural network trained on Type 1b models and tested on Type 1b models having the same structure. (a), (d), and (g) are the testing velocity models. (b), (e) and (h) are the corresponding predicted models. (c), (f) and (i) show the cross-section through each of the corresponding models at the dashed line (pseudo-well locations). The same color bar is used for all velocity models.

Figure 2.10 Testing results of the neural network trained on Type 1a models and tested on Type 1b models. (a), (d), and (g) are the testing velocity models. (b), (e) and (h) are the corresponding predicted models. (c), (f) and (i) show the cross-section through each of the corresponding models at the dashed line (pseudo-well locations). The same color bar is used for all velocity models.
Figure 2.11  Histograms showing the normalized rms error for the networks trained on (a) Type 1a, (b) Type 1b, (c) Type 2a, and (d) Type 2b data and tested on Type 1b data.

Figure 2.12  Testing results of the neural network trained on Type 2b models and tested on Type 1b models. (a), (d), and (g) are the testing velocity models. (b), (e) and (h) are the corresponding predicted models. (c), (f), (i) show the cross-section through each of the corresponding models at the dashed line (pseudo-well locations). The same color bar is used for all velocity models.

Figure 2.13  Testing results of the neural network trained on Type 2a models and tested on Type 1b models. (a), (d), and (g) are the testing velocity models. (b), (e) and (h) are the corresponding predicted models. (c), (f) and (i) show the cross-section through each of the corresponding models at the dashed line (pseudo-well locations). The same color bar is used for all velocity models.

Figure 2.14  Testing results of the neural networks trained on Type 2b models with square sizes ranging between (b) and (h) $1 - 25$, (c) and (i) $5 - 25$, (d) and (j) $10 - 25$, and (e) and (k) $15 - 25$ grid points. (f) and (l) present the cross-sections for the recovered models of the first testing model (a). (f) and (l) present the cross-sections at the dashed lines (pseudo-well locations) for the recovered models of the first and second testing models, respectively. The same color bar is used for all velocity models.

Figure 2.15  Testing results of the neural networks trained on Type 2b models with square sizes ranging between (b) and (h) $1 - 5$, (c) and (i) $1 - 10$, (d) and (j) $1 - 15$, and (e) and (k) $1 - 25$ grid points. (f) and (l) present the cross-sections at the dashed lines (pseudo-well locations) for the recovered models of the first and second testing models, respectively. The same color bar is used for all velocity models.

Figure 2.16  The first Marmousi model subset used to test the four trained networks. (a) Marmousi model subset. Prediction results for (b) Type 1b, (c) Type 1a, (e) Type 2b, and (f) Type 2a training. (d) and (g) present the cross-sections corresponding to Types 1a/1b and 2a/2b, respectively. The same color bar is used for all velocity models.

Figure 2.17  The second Marmousi model subset used to test the four trained networks. (a) Marmousi model subset. Prediction results for (b) Type 1b, (c) Type 1a, (e) Type 2b, and (f) Type 2a training. (d) and (g) present the cross-sections corresponding to Types 1a/1b and 2a/2b, respectively. The same color bar is used for all velocity models.

Figure 2.18  The third Marmousi model subset used to test the four trained networks. (a) Marmousi model subset. Prediction results for (b) Type 1b, (c) Type 1a, (e) Type 2b, and (f) Type 2a training. (d) and (g) present the cross-sections corresponding to Types 1a/1b and 2a/2b, respectively. The same color bar is used for all velocity models.

Figure 2.19  Histograms showing the normalized rms error for the networks trained on (a) Type 1a, (b) Type 1b, (c) Type 2a, and (d) Type 2b data and tested on 600 subsets of the Marmousi model.

Figure 2.20  Comparison of the magnitude and phase components of the data simulated using Type 1b/1a models. (a) True model. (b) Type 1b result. (c) Magnitude and phase comparisons for data modeled using result in (b) for 3 Hz, 18 Hz and 33 Hz. (d) Type 1a results. (d) Type 1a result. (e) Magnitude and phase comparisons for data modeled using result in (d) for 3 Hz, 18 Hz and 33 Hz.
Figure 2.21 Comparison of the magnitude and phase components of the data simulated using Type 2b/2a models. (a) True model. (b) Type 2b result. (c) Magnitude and phase comparisons for data modeled using result in (b) for 3 Hz, 18 Hz and 33 Hz. (d) Type 2a results. (e) Type 2a result. (f) Magnitude and phase comparisons for data modeled using result in (d) for 3 Hz, 18 Hz and 33 Hz.

Figure 2.22 Histograms showing the normalized data-fitting rms error. The left, center and right columns are for 3 Hz, 18 Hz and 32 Hz data, respectively. (a)-(c) present results for Type 1a data, (d)-(f) for Type 1b data, (g)-(i) for Type 2a data, and (j)-(l) for Type 2b data.

Figure 3.1 16 representative 2.0 km × 2.0 km synthetic velocity models generated for network training.

Figure 3.2 Diagrams illustrating two types of RNNs. (a) A one-to-one RNN, (b) a simplified representation of a one-to-one RNN where the recurrence arrow is replaced by its equivalent (the hidden-state vector), and (c) the many-to-many RNN network used in this work.

Figure 3.3 In a feed-forward neural network (a) each image is fed independently, leading the network to recognize the object in each image as an airplane. However, in a recurrent neural network (b) the hidden-state vector acts as the network memory so that as we inject the last image in the sequence, information about the previous images had been stored in the hidden-state vector. This leads the network to recognize the movement of the object in the images as an airplane taking off.

Figure 3.4 Three examples of the 2D Gaussian smoothing operator used in this work. When the Gaussian smoothing radius ($\sigma_S$) is 18, the smoothing operator is large and the smoothed velocity model only has the general trend of the true velocity model. As we reduce $\sigma_S$, the smoothing operator becomes smaller and more velocity structures are preserved in the smoothed model. Large filters are used for smaller frequencies, while smaller filters are used for higher frequencies.

Figure 3.5 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.

Figure 3.6 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 $h_i$ and $c_i$ 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.

Figure 3.7 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.
Figure 3.8  Training (green) and validation (red) learning curves for the LSTM-RNN used in this experiment. ................................................................. 57

Figure 3.9  (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. ... 58

Figure 3.10  Four examples of velocity models estimated by the trained LRTM-RNN network. (a), (d), (g), and (j) show the true testing velocity models. (b), (e), (h), and (k) present 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 colorbar is used for all velocity models. The normalized root-mean-square (NRMS) misfit between the true and predicted models is 0.023 (b), 0.035 (e), 0.055 (h), and 0.079 (k). These misfit values fall in the 82th, 67th, 42nd, and 22nd percentiles of the NRMS misfit histogram for all 5000 testing data, respectively. .................. 59

Figure 3.11  A histogram showing the NRMS misfit between the true and recovered 9.9 Hz velocity models calculated using equation 3.1. ................................................................. 60

Figure 3.12  Comparison of the amplitude and phase components of the data simulated using the LSTM-RNN estimated models. (a) True model. (b) Cross-sections through the true and recovered models at the location of the dashed line in (a). (c) Recovered model at 9.9 Hz. (d) Magnitude and (e) unwrapped phase comparisons for data modeled using the models in (a) and (c) at 2.2 Hz, 5.5 Hz and 9.9 Hz. The normalized root-mean-square (NRMS) misfit between the true and modeled data at 9.9 Hz is 0.041 and 0.003 for the amplitude and phase, respectively. These misfit values fall in the 79th and 99th percentiles, respectively, of the NRMS misfit histogram for all 5000 testing data. ................................................................. 61

Figure 3.13  Comparison of the amplitude and phase components of the data simulated using the LSTM-RNN estimated models. (a) True model. (b) Cross-sections through the true and recovered models at the location of the dashed line in (a). (c) Recovered model at 9.9 Hz. (d) Magnitude and (e) unwrapped phase comparisons for data modeled using the models in (a) and (c) at 2.2 Hz, 5.5 Hz and 9.9 Hz. The normalized root-mean-square (NRMS) misfit between the true and modeled data at 9.9 Hz is 0.039 and 0.005 for the amplitude and phase, respectively. These misfit values fall in the 82nd and 97th percentiles, respectively, of the NRMS misfit histogram for all 5000 testing data. ................................................................. 62

Figure 3.14  Comparison of the amplitude and phase components of the data simulated using the LSTM-RNN estimated models. (a) True model. (b) Cross-sections through the true and recovered models at the location of the dashed line in (a). (c) Recovered model at 9.9 Hz. (d) Magnitude and (e) unwrapped phase comparisons for data modeled using the models in (a) and (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. The normalized root-mean-square (NRMS) misfit between the true and modeled data at 9.9 Hz is 0.035 and 0.161 for the amplitude and phase, respectively. These misfit values fall in the 89th and 1st percentiles, respectively, of the NRMS misfit histogram for all 5000 testing data. ........ 63
Figure 3.15 Comparison of the amplitude and phase components of the data simulated using the LSTM-RNN estimated models. (a) True model. (b) Cross-sections through the true and recovered models at the location of the dashed line in (a). (c) Recovered model at 9.9 Hz. (d) Magnitude and (e) unwrapped phase comparisons for data modeled using the models in (a) and (c) at 2.2 Hz, 5.5 Hz and 9.9 Hz. The normalized root-mean-square (NRMS) misfit between the true and modeled data at 9.9 Hz is 0.127 and 0.051 for the amplitude and phase, respectively. These misfit values fall in the 11th and 44th percentiles, respectively, of the NRMS misfit histogram for all 5000 testing data.

Figure 3.16 Histograms showing the NRMS misfit between data modeled through the true and 9.9 Hz recovered model calculated using equation 3.1. (a) Magnitude. (b) Unwrapped phase.

Figure 4.1 Representative 1.25 km × 4.0 km examples drawn from the three training model classes. (a) Geological models with faulted layered structures. (b) Purely geometric models at different scales with variably oriented structures. (c) Background models with either a constant value or a gradient in a uniformly distributed random orientation.

Figure 4.2 Illustration of the five steps used to generate a three-layer model. (a) The control points of the first interface are randomly specified followed by (b) a linear piece-wise interpolation between these points. (c) The control points of the second interface are specified followed by (d) a linear interpolation. (e) 1-D Gaussian smoothing operator is applied to the constructed interfaces. (f) Velocity values are assigned to the resulting layers.

Figure 4.3 Illustration of the model faulting process. (a) A fault line is specified by interpolating a straight line between two randomly located points at the top and bottom of model boundary. (b) Layers are shifted at the fault line by a randomly specified amount. (c) A model with three parallel fault line. (d) The layers are shifted at the fault line locations by the same vector offset.

Figure 4.4 Comparison between conventional and convolutional RNNs. (a) The two fully connected layers comprising the conventional RNN. Information flows between step by adding the output of the hidden-state layer to that of the layer handling the input data. (b) The same process as in (a) but with convolutional operators connecting the input and output instead of connective weights. Items with the same role are indicated with the same color.

Figure 4.5 Comparison between single (a) conventional RNN and (b) and LSTM cells that shows how information flows from previous step $t - 1$ is processed at the current $t$ step. Each green box represents a layer with the accompanying text indicating the associated activation function. $x_t$ is the input at the current step, $h_{t-1}$ is the hidden state vector flowing from the previous step, and $h_t$ is the hidden-state vector produced by processing $x_t$ and $h_{t-1}$ using the operations in the cell.

Figure 4.6 A sketch showing the network architecture used in this work. The network consists of three ConvLSTM layers. Information flows from one frequency to the next via two memory vectors. Each input frequency is constrained by the corresponding 2-D Gaussian-smoothed velocity model.
Figure 4.7 Convolutional parameters for each network layer. The input data frequency slice consists 38 sources and 400 receivers with the top and bottom 38 rows consisting of real and imaginary data components. The first two numbers below each ConvLSTM layer represent the 2-D convolutional filter size in the vertical and horizontal directions, the last number is the number of filters, and their product is the number of output feature maps for each layer. The output 2-D feature maps for ConvLSTM3 are flattened into a vector for connection to the fully connected output velocity model layer. 77

Figure 4.8 Illustration of the SSIM loss function used for network training. The SSIM performs a window-to-window (instead of point-to-point) comparison between true and predicted velocity models. The squares in each image represent the decreasing window size used to measure the fit between true and predicted velocity models with increasing frequencies. 78

Figure 4.9 Training and validation curves for Networks (a) $G$ and (b) $H$. 79

Figure 4.10 The BP2004 model with white and purple rectangles showing the respective locations of horizontally and vertically adjacent model subsets. 80

Figure 4.11 The 30 BP 2004 model subsets recovered using network $G$ patched into their true locations (a) before and (b) after smoothing. 81

Figure 4.12 The 30 BP 2004 model subsets recovered using network $H$ patched into their true locations (a) before and (b) after smoothing. 82

Figure 4.13 Comparison between models recovered by Networks $G$ and $H$ for subsets indicated by the white rectangles in (a). The true velocity models extracted at the location indicated by the (b) solid and (c) dotted lines. (d) and (e) The two subsets recovered using Network $G$. (f) and (g) The two subsets recovered using Network $H$. 82

Figure 4.14 Comparison between models recovered by Networks $G$ and $H$ for subsets indicated by the white rectangles in (a). The true velocity models extracted at the location indicated by the (b) solid and (c) dotted lines. (d) and (e) Subsets recovered using Network $G$. (f) and (g) Subsets recovered using Network $H$. 84

Figure 4.15 Comparison between models recovered by Networks $G$ and $H$ for subsets indicated by the white rectangles in (a). The true velocity models extracted at the location indicated by the (b) solid and (c) dotted lines. (d) and (e) Subsets recovered using Network $G$. (f) and (g) Subsets recovered using Network $H$. 85

Figure 4.16 The SSIM evaluation metric computed for 2200 different BP2004 model subsets where values of 1.0 and 0.0 represent exact similarity and fully uncorrelated statistics, respectively. (a) Network $G$ and (b) Network $H$. 86

Figure A.1 A screenshot granting permission from Geophysics. 96

Figure A.2 A screenshot granting permission from IEEE. 97
LIST OF ABBREVIATIONS

Adaptive Moment Estimation ................................................ ADAM
Artificial Neural Network .................................................. ANN
Convolutional Long Short-Term Memory .............................. ConvLSTM
Convolutional Neural Network ........................................... CNN
Finite Difference ............................................................... FD
Full-Waveform Inversion .................................................... FWI
Generative Adversarial Network ......................................... GAN
Graphics Processing Unit ................................................... GPU
Long Short-Term Memory ................................................... LSTM
Machine Learning ............................................................. ML
Mean Squared Error .......................................................... MSE
Normalized Root Mean Square .......................................... NRMS
Out of Distribution .......................................................... OOD
Rectified Linear Unit ........................................................ ReLU
Recurrent Neural Network ................................................ RNN
Root Mean Square ........................................................... RMS
Structural Similarity Index Measure ................................... SSIM
ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my adviser Dr. Jeffrey Shragge for his support, patience, and motivation throughout my research. Working with Jeff was a learning journey that was also quite enjoyable. I’ve acquired much insight from him in both technical and non-technical matters. Throughout my research journey, Jeff has been consistently understanding and supportive, creating an ideal atmosphere for my studies.

I express my warm thanks to my thesis committee members Dr. Paul Sava, Dr. Douglas Nychka, Dr. Yaoguo Li, and Dr. Zane Jobe for kindly serving on my committee and for their valuable comments on my thesis. I would like to also thank the rest of the CWP faculty Dr. Ilya Tsvankin and Dr. Roel Sneider for their help and support with different aspects of my research and my life in Mines in general. I would also like to extend my gratitude to the faculty members of the CWP for cultivating a family-like atmosphere for the group’s students.

I would like to thank my C-Team and CWP colleagues for their valuable insights into my research projects. Special thanks to my CWP colleague and friend Khalid Almuteri for the fruitful discussions that helped me navigate various challenges in my research.

I extend my deepest gratitude to my parents for their unwavering motivation and support not only throughout this project, but also at every stage of my life. Their love and support has been the main driving force behind my determination and perseverance. Sincere thanks to my whole family, for their encouragement and support.

I wish to convey my heartfelt gratitude to my wife, who has been my steadfast pillar during these challenging years. Her unwavering belief in me, often encapsulated in her encouragement of ‘you can do it,’ has been invaluable to my journey. Her celebratory spirit after each of my milestones greatly fueled my drive for success.

Lastly, I would like to thank my sponsoring company (Saudi Aramco) for their financial support.
Seismic waves are commonly used as a remote sensing tool to investigate the earth’s interior. The seismic imaging problem is routinely divided into two separate steps: (1) estimating a smooth background velocity model, and (2) using this model in migration applications to locate the sharp subsurface discontinuities. The earlier days of seismic exploration used ray-based travel-time tomography methods [1] for estimating the background velocity model; however, the past few decades have seen the rise of wave-equation migration velocity analysis (MVA) methods [2–6] that are based on computing numerical solutions of the governing acoustic (or elastic) wave equation. Although these methods have been largely successful in stratified media characterized by weak lateral heterogeneity (i.e., $v \approx v(z)$ profiles), these methods faced serious limitations when encountering complex velocity models (i.e., $v = v(x, y, z)$ profiles).

The aforementioned velocity model building limitations motivated the development and refinement of full waveform inversion (FWI), which is the process of obtaining a high-resolution subsurface velocity model that minimizes the misfit between observed and forward modeled seismic data [7, 8]. FWI uses the full waveform (i.e., phase and magnitude), not just the kinematics (travel time) as in most traveltime tomography applications, to recover absolute estimates of earth model parameters used in ensuing seismic interpretation, reservoir characterization, and direct hydrocarbon detection analyses. However, there are two main challenges of FWI. The first and fundamental challenge is that FWI is highly non-linear because the governing acoustic (or elastic) wave equation is itself non-linear with respect to model parameters. This leads to non-convex objective functions with numerous local minima, a factor which routinely prevents gradient-descent methods from successfully converging to the global minimum [9]. A second more practical issue is that FWI is computationally expensive due to the need to perform 3-D full-wavefield simulation for hundreds or thousands of sources per FWI iteration through a model that may have millions to billions of grid points.

The severe non-linearity of FWI is usually manifest in the phenomenon of cycle skipping. Due to the oscillatory nature of seismic waveforms, if the initial earth model produces simulated seismic data that are more than half a period shifted from the observed data, then the optimization algorithm mistakenly matches cyclical waveforms, which causes the model to converge to a local minimum that may be far away from the global one. Thus, an initial model that simulates data to within half a cycle of the observed data is a precondition for accurate FWI implementation. Unfortunately, such models are challenging to produce and rarely available in realistic seismic exploration situations.
Because the kinematics of wave propagation are largely governed by the low-wavenumber (long-wavelength) velocity model components, the FWI process benefits by recovering these components first to move the inversion to the vicinity of the global minimum. Sirgue and Pratt [10] show that the low-wavenumber velocity model components can be recovered from data using either low frequencies or far source-receiver offsets. This observation prompted the development of a multi-scale FWI approach, where the lowest frequencies available in the data are inverted first to recover the longer wavelength model components, followed by higher frequencies to infill shorter wavelength detail [11]. Nevertheless, in the absence of an accurate initial model, frequencies as low as 1 or 2 Hz may be required to successfully guide the inversion process in the first few iterations. Unfortunately, frequencies in this range are difficult to stimulate and record with a high signal-to-noise ratio due to mechanical limitations of land or marine seismic sources. The FWI non-linearity problem has been addressed extensively in the literature with proposed solutions that include different methods for computing the misfit function, model regularization, artificial low-frequency data generation, and gradient conditioning.

1.1 Enter Machine Learning

The resurgence of machine learning (ML) techniques in recent years, driven largely by advances in GPU computing, is introducing new paradigms for automated velocity model building that do not inherently suffer from the aforementioned challenges. Artificial neural networks (ANNs) have been applied successfully in a variety of seismic applications, including seismic processing [12], automated first-break [13] and velocity picking [14, 15], least-squares migration (LSM) [16], amplitude variation with offset (AVO) inversion [17], automated horizon and fault interpretation [18–20], and seismic data interpolation [21]. More importantly in this context, ANNs have been used to either augment FWI to reduce the non-linearity issues and speed up convergence [22–26], or to perform the inversion themselves [27–31].

Neural networks consist of an input layer, one or more hidden layers, and an output layer. The input layer receives the input data, and the hidden layers process the input data through a series of mathematical operations (often involving weights and biases) to produce a prediction at the output layer. During the NN training process, training data are input into the network and processed through a forward pass to produce the corresponding network outputs. The difference between the true and predicted outputs then can be computed using a predefined loss (i.e., objective) function, and the network weights are updated using a backpropagation process [32]. Due to the loss function being highly non-convex, using a fixed learning rate (i.e., step length) slows convergence and carries the risk of becoming stuck in a local minimum. To speed up convergence to the global minimum, the adaptive moment estimation (ADAM) algorithm is often used. ADAM adjusts the learning rate for each parameter based on previous gradient
step information. It maintains two running averages for each weight: one for the gradients themselves and another for their squared values. The first average gives ADAM a sense of the direction and magnitude of the recent gradients, acting like a momentum, pushing updates in consistent directions. The second average helps ADAM adjust the size of the updates based on the historical variability of each gradient.

After each training epoch (i.e., iteration), the network is tested using a set of input-output pairs, termed validation data, that were not used in the training process. This step serves as a test of whether the reduction in the training loss function value is the result of the network learning the relationship between the inputs and outputs, or merely memorizing them. When training and validation losses both decrease, this indicates that the network is actually “learning” the relationship between input and output pairs. However, when the training loss decreases while the validation loss remains the same or is increasing, this indicates that the network is merely “memorizing” the input-output relationships. That is, the validation data test effectively evaluates the network’s ability to generalize. Network generalization is defined as the ability of a trained neural network to make accurate predictions on as yet unseen data. Thus, generalization is a key ML goal because it enables models to make accurate predictions on data not before seen, which is essential for their usefulness in real-world applications.

1.2 Research Challenge

For the seismic velocity model building problem, the challenge of generalization translates to whether the trained network is able to recover models not used in the training process. Seismic velocity model building is usually implemented using a neural network architecture that operate in an end-to-end manner, meaning that it receives seismic data as its input and constructs a velocity model as its output in a single step. This is in contrast with deterministic approaches where the final velocity model is generated through a series of steps that require iterating on an initial velocity model estimate. In addition to following the end-to-end principle, these approaches typically rely on supervised learning, which requires training data containing both velocity models and their corresponding seismic data. Selecting appropriate training models is among the most critical factors that influence the network’s generalization ability.

The task of generalization becomes even more difficult in velocity model building applications because seismic data often are acquired in areas where the geology is largely unknown, and designing training models requires knowledge about the subsurface geology in the area of interest. Because that is not possible in some instances, a network must adequately generalize to out-of-distribution (OOD) testing models for it to be usable in realistic situations. That is, the network needs to recover accurate velocity models even if the testing model has a velocity distribution different than those in the training data set.
In most published work on this topic, the trained networks succeeded in recovering testing models not used in the training process. However, the testing models were generated using the same engine that generated the training data, which means they have a similar velocity distribution as well as the type of geological structures as the training data. The work presented in this thesis extends the definition of generalization to the more realistic case of the trained network being able to recover subsurface models that have an unknown (but bounded) velocity distribution and may have structures that are drastically different from those found in the training models.

There are two common themes in the majority of the publications that apply NNs to tackle the seismic velocity model building problem. First, the numerical simulations and analysis are performed in the time domain and thus require entire seismic shot gathers to be fed simultaneously into the network. This often results in a input data volumes that place strong restrictions on the experimental model size due to GPU memory hardware limitations. This problem is usually mitigated by feeding the shot gathers to a series of encoding layers that transform the large input data to a much smaller latent vector that encompasses what the network finds to be the most important input data features. This is usually followed by a series of decoding layers that transforms the latent vector into a velocity model. While it reduces the input data size, the encoding process does not give the user the option to separate different data scales whether by frequency content or by source-receiver offset. It encapsulates the burden of defining the most important features in the data to a black box, represented through the encoding layers. Therefore, it does not lend itself to the strategies usually used to mitigate FWI non-linearity, chief among which is the multi-scale approach.

The second common theme is that, while sharing an autoencoder network architecture with some variations, they address the generalization problem in one of two ways. The first approach is to incorporate physics into the neural network dynamics by including some form of the wave equation in the loss function [25, 33]. The second approach focuses on building “geologically realistic” velocity models of the subsurface [34, 35]. The former usually increases network training costs, sometimes to the equivalent of a single FWI iteration per training epoch. The latter assumes prior knowledge of expected subsurface structures and velocity distribution in the area of interest, an assumption that does not hold where insufficient geological information is known to construct such models. There has been no investigations of how increasing the variability of structures in training data by augmenting the geological structures with, e.g., purely geometric shapes would affect the network generalization.

To address the issue of the input data size, I develop a novel frequency-stepping NN approach that exploits the multi-scale FWI strategy. The input data consist of frequency slices, as opposed to time-domain shot gathers. In a sense, applying the Fourier transform and extracting a discrete number of
frequency slices can be viewed as performing a job similar to that of an encoding network - though doing so in a manner fully transparent to the user. In general, the Fourier transform produces an output with the same number of parameters as its input. Nevertheless, in practice, a limited number of output frequency slices are needed to recover an accurate subsurface velocity model [36]. Thus, there are three important advantages of the developed frequency-domain approach over the encoding-decoding strategy: (1) it is more memory and computationally efficient; (2) the input frequency slices have a physical meaning, as opposed to the encoded latent vector; and (3) the user has the power to select what frequencies to use and in which order.

To address the issue of network generalization, I develop a novel approach that combines different classes of training models to increase training data variability. Restricting the training models to a certain set of geological structures also limits the network’s ability to recover velocity models of other types. Regardless of the geological realism of the training models, the network will be unable to recover structures not explicitly included in the training data. Conversely, the training model variability can be increased by augmenting geological models with other model classes constructed from randomly generated velocity structures. This would improve the network’s ability to recover velocity models that cannot be predicted during training, which is the case in areas with unknown geology.

1.3 Thesis Outline

The thesis is structured into three technical chapters, two of which have been published in peer-reviewed journals, and one of which will be submitted for publication. As the student first author of the following technical chapters, I developed the theory and performed the numerical computations. As the thesis advisor and co-author, Dr. Jeffrey Shragge supervised the analytical and numerical findings. Both authors discussed the results and equally contributed to writing the manuscripts and documenting the research work.

Chapter 2 explores the feasibility of using purely geometrical velocity models to train a convolutional neural network (CNN) to solve the seismic velocity model building problem. I train two 1D CNNs with the same hyperparameters but different training data. One network was trained using geological training models, not dissimilar to those commonly used in training the vast majority of neural networks in the literature; the other was trained using purely geometrical training models. The geometrical training models consist of squares with randomly assigned velocities that were not prescribed to follow any coherent structure. The network trained using the geometrical shapes recovered a wide range of geological structures, even those with drastically different properties than found in its training data set. In addition, despite their rudimentary structures, geometrical training data showed slightly improved generalization
capabilities than geological training data because they were not biased toward any specific geological structures. This analysis demonstrates the potential benefit of using geometrical shapes in the training data set to improve generalization. This chapter has been published in Geophysics:


Chapter 3 develops a novel approach to building seismic velocity models using neural networks. Instead of the prevalent approach of using time-domain shot gathers as the network input, followed by an encoding layer to reduce its size, I use discrete frequency slices generated by applying a Fourier Transform to the input data set and extracting a subset of available frequencies. In addition to drastically reducing the size of the input data, this approach allows for exploiting the multi-scale strategy widely used to mitigate FWI non-linearity challenges. I develop a frequency-stepping velocity model building approach 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 or chosen, while the corresponding output sequences are frequency-dependent smoothed velocity models. Qualitative and quantitative analysis show that the proposed approach recovers accurate velocity models, with the recovered model resolution improving with increasing input frequency. This chapter has been published in IEEE Transactions on Geoscience and Remote Sensing:


Chapter 4 aims to significantly improve the generalization capabilities of the frequency-stepping neural network by proposing a novel hybrid training data approach that consists of a combination of geological and purely geometric training data. As opposed to the squares-only models used in Chapter 2, I use a more elaborate process to construct continuous random geometric shapes. In addition, I improve the frequency-stepping network by replacing the fully connected layers constituting the LSTM cell with convolutional layers. In addition to being more computationally efficient, convolutional RNNs preserve the spatial distribution of the different input receiver data. That is, the input data can be arranged as a 2-D array, with each shot data occupying a row, as opposed to flattening the 2-D array into a 1-D vector as is done in fully connected RNNs. To demonstrate the potential for improved generalization when using hybrid training data sets, I compare the model estimates results from two trained networks: one using a
hybrid set of models, the other with only geological models. Next, I test the two networks using subsets extract from the BP2004 benchmark model [37] with complex salt structures that are fully absent from the models used in the training process. Qualitative and quantitative analyses show that models recovered using hybrid training data are significantly more accurate than those recovered using geological training data alone, with arbitrarily shaped salt bodies being accurately delineated by the trained hybrid network.

Chapter 5 presents the general conclusions of the thesis along with a discussion of future work to further improve the network applicability to field data.
CHAPTER 2
SEISMIC VELOCITY MODEL BUILDING USING NEURAL NETWORKS: TRAINING DATA DESIGN AND LEARNING GENERALIZATION

2.1 Abstract

Data-driven artificial neural networks (ANNs) 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 high-resolution subsurface velocity models. However, a significant challenge of ANNs is training generalization, which is the ability of an ANN to apply the learning from the training process to test data not previously encountered. In the context of velocity model building, this means learning the relationship between velocity models and the corresponding seismic data from a set of training data, and then using acquired seismic data to accurately estimate unknown velocity models. We ask the following question: what type of velocity model structures need be included in the training process so that the trained ANN can invert seismic data from a different (hypothetical) geological setting? To address this question, we create four sets of training models: geologically inspired and purely geometrical, both with and without background velocity gradients. We find that using geologically inspired training data produce models with well-delineated layer interfaces and fewer intra-layer velocity variations. The absence of a certain geological structure in training models, though, hinders the ANN’s ability to recover it in the testing data. We use purely geometric training models consisting of square blocks of varying size to demonstrate the ability of ANNs to recover reasonable approximations of flat, dipping, and curved interfaces. However, the predicted models suffer from intra-layer velocity variations and non-physical artifacts. Overall, the results successfully demonstrate the use of ANNs in recovering accurate velocity model estimates, and highlight the possibility of using such an approach for the generalized seismic velocity inversion problem.

2.2 Introduction

One of the biggest challenges in seismic exploration is building a velocity model that minimizes the misfit between observed and forward-modeled seismic data. Full waveform inversion (FWI) [7, 8] has been one of the most successful deterministic approaches for estimating high-resolution models. However, because the governing wave equation is non-linear with respect to medium parameters, FWI is characterized by non-convex objective functions with numerous local minima that frequently forestall convergence of optimization-based methods toward the globally minimum solution [9].
An interesting alternative to deterministic velocity model building approaches is to use artificial neural networks (ANNs). ANNs have been successfully applied in a variety of seismic applications, including seismic processing [12], automated first-break [13] and velocity [14, 15] picking, least-squares migration (LSM) [16], amplitude variation with offset (AVO) inversion [17], automated horizon and fault interpretation [18–20], seismic data interpolation [21]. Other authors succeeded in using a combination of FWI and ANNs to invert seismic data [22–24]. Additionally, ANNs have been used to generate solutions of wave equations, which serve as the modeling engine for imaging and inversion problems. [38] implement neural networks for simulating seismic wavefields by solving the time-domain acoustic wave equation based on training data sets generated through deterministic forward modeling of random \( v(z) \) velocity models. [39] show examples of an ANN-based velocity inversion of semblance cubes based on training examples generated through curvilinear block models. [40] develop a 1D NN-based inversion methodology and successfully apply it to estimate velocity models from 2D field seismic data.

While ANNs do not inherently suffer from challenges commonly associated with deterministic methods, they have their own set of challenges. Foremost among them is training generalization, which is the ability of a neural network to perform well when tested on data features not included in the training data set. An important observation in previous applications of ANNs to the seismic inversion problem is that the data sets used for the training and testing processes often closely resemble each other. While this may make sense intuitively, an important question is how close do they have to be to achieve a successful testing validation and accurate ensuing predictions on independent data sets with different velocity model characteristics (e.g., layer geometry, velocity structures, geobodies, faults)? Specifically for the seismic velocity inversion problem, how close must the velocity models used for training and testing be to still recover a reasonable velocity model prediction? This leads to a number of questions regarding the ANN-based seismic velocity inversion training process: (1) Should the velocity models used for training be highly “geological” (e.g., with plausible stratigraphic, structural, and petrophysical attributes), which can be challenging to construct? (2) Can one use easy-to-define geometric objects (e.g., squares with different size) for training purposes and still realize acceptable inversion results? or (3) Does one need to explicitly include background velocity gradients in the training models to capture typical long-wavelength velocity variations? In essence, how much geological and physical realism needs to be included in the training velocity models to guarantee or at least maximize the probability of converging to an acceptable velocity model?

In this work, we investigate the ability of a certain neural network architecture to recover different model classes through seismic inversion. We analyze how the features in a certain set of training velocity models are imprinted in the structure of the recovered model estimates. To our knowledge, there has been
limited published work that analyzes the ability of neural networks to predict velocity models with structures that differ greatly from those found in the training models. This problem is one example of training generalization, which is one of the biggest challenges in the general field of machine learning and more specifically in scientific applications. Therefore, understanding how the structure and scale length of the selected training model features affect the accuracy and resolution of the predicted models is crucial before ANNs can be used more broadly for seismic velocity inversion purposes.

We begin by introducing the basics of the seismic forward problem to show our approach for generating training data. We then develop different classes of training/testing data and explain how they are used in our numerical experiments. Next, we present an overview of ANNs and, in particular, of convolutional neural networks (CNNs). We subsequently explain how we use them to invert seismic data to estimate velocity models. Finally, we present a series of numerical experiments designed to evaluate and better understand the relationship between each input class of training models and the predicted output models.

2.3 Generating Training Data

Training an ANN requires training data that contain input-output pairs. Here, the input data consist of surface seismic waveforms, while the outputs are the corresponding velocity models. To generate seismic waveforms, we solve the 2D constant-density acoustic wave equation in the frequency domain using an implicit finite-difference (FD) scheme.

To train a neural network, we need to have a sufficient number of training data, where “sufficiency” varies significantly depending on the application. In the experiments described below, we use synthetic velocity models and corresponding seismic data to train the ANN. To assess the sensitivity of neural networks to the velocity model structure used in the training process, we synthesized four different classes of P-wave velocity models with $V_p$ values ranging between 1.5 km/s and 4.0 km/s.

2.3.1 Semi-Geological Models - Types 1a/1b

The first class of training velocity models coarsely mimics subsurface geologic structures. We term these “semi-geological” because they are constructed using purely mathematical operations, but have layer-based outputs constrained to avoid obvious geologically infeasible structures. We construct each velocity model in a number of procedural steps. First, we specify a random number of layers and construct each surface using a sequence of randomly generated numbers, including the number and locations of control points defining the surface topology. Each randomly generated control point is constrained to avoid obvious non-physical layer behavior and to ensure layer thicknesses do not become thinner than a pre-specified value. We then use cubic splines to construct layer interfaces by interpolating between the control points.
and extrapolating to the model boundaries.

The area between the constructed interface and the overlaying layer (or the free surface) is populated with velocity values that depend on numerous factors that are set at different levels. At the data set level, we specify whether the current model belongs to “Type 1a” with a constant velocity value within each layer or “Type 1b” where randomly selected layers are permitted to have velocity gradients. At the individual model level, we randomly decide whether model velocities strictly increase with depth or whether velocity inversions are permitted. Finally, at the layer level for Type 1b models, we randomly choose whether the velocity of a given layer should be constant or have a linear gradient in one or both directions. These steps are repeated until all layers are constructed. Figure 2.1(a) shows members of the Type 1a data set where each model layer has a constant velocity, while Figure 2.1(b) shows Type 1b examples where some layers have linear velocity gradients.

2.3.2 Squares-Only Velocity Models - Types 2a/2b

The second class of training velocity models is purely based on a simple geometric shape, where we made no attempt to model geologic-like structures. Each velocity model consists of squares of the same size, except when truncated at model boundaries. The first step in constructing a “squares-only” velocity model is to randomly specify the square size, which we restrict to be between 1 and 25 grid points. Each square is then assigned a distinct velocity value. Similar to the Type 1a/1b models, the velocity values could be uniform over the entire square (“Type 2a” models) or varying gradually in one or both directions (“Type 2b” models). Figure 2.2(a) shows members of the Type 2a data set where each square in each model is filled with a constant velocity; Figure 2.2(b) presents Type 2b examples where some squares have linear velocity gradients in one or both directions.

2.3.3 Data Set Characteristics

For each of the model classes described above, all generated random numbers used in the velocity model building process were drawn from a uniform distribution, where the minimum and maximum values are fixed a priori. For training purposes, we generated 10,000 velocity models for each model type. We also generated 2000 and 10,000 Type 1b velocity models for initial validation and subsequent testing, respectively. We validate and test our neural networks using only Type 1b models because we are interested in constructing more realistic layered models with velocity gradients. All models are of size $n_x \times n_z = 50 \times 50$ grid points at a 10 m grid spacing. We chose these model dimensions mainly to increase the efficiency of generating velocity models, synthesizing seismic data, and training and testing each of the four ANNs. The extension of the methods developed here to larger models should be straightforward in
Figure 2.1 A number of members from the two sets of randomly generated “semi-geological” velocity models, along with corresponding “pseudo-well” cross-sections extracted at the dashed line of each model. Examples of (a) Type 1a velocity models with no velocity gradients and (b) Type 1b velocity models where some randomly selected layers have velocity gradients.
Figure 2.2 A number of members from the two sets of randomly generated “semi-geological” velocity models, along with corresponding “pseudo-well” cross-sections extracted at the dashed line of each model. Examples of (a) Type 1a velocity models with no velocity gradients and (b) Type 1b velocity models where some randomly selected layers have velocity gradients.
principle, but becomes computationally challenging due to hardware limitations (i.e., limited global memory of GPU cards). When modeling the frequency-domain shot gathers, we use the sources and receivers situated on the grid every 20 m and 10 m, respectively. The fixed source and receiver arrays were situated at a depth of 20 m. We used a 20 Hz Ricker wavelet and simulated seismic wavefields using the above approach for 30 frequencies from 3 Hz to 32 Hz at 1 Hz increments. The complex data components were saved for the training and testing procedures discussed below.

2.4 Artificial Neural Networks

ANNs consists of cells that are grouped in different ways to form hidden, input and output layers. The cells of the hidden layers are connected to all of those in the previous/ ensuing layer via weights, which determine how information flows between the input and output layers. Neurons in the input and output layers only have outgoing and incoming connections, respectively. Figure 2.3(a) shows an ANN with a single hidden layer as well as the connections between the input layer and the first neuron in the hidden layer. This diagram indicates that the first neuron in the hidden layer is influenced by all the neurons in the previous layer as well as the connective weights. If a neural network has more than one hidden layer (i.e., between the input and output layers), it is usually referred to as a deep neural network where the depth is defined by the number of hidden network layers. Because deeper ANNs can learn features at different levels of abstraction [41], more complex non-linear functions need additional hidden layers to approximate them.

The first step in performing a forward run of a neural network is populating each cell in the input layer with an initial data value. One then evaluates the value at each cell in the first hidden layer in three steps: (1) compute the weighted sum of all the values in the previous layer (the weights being a trained parameter), (2) add a bias (another trained parameter), and (3) pass the result through a non-linear “activation” function to introduce non-linearity into the system. The bias shifts the activation function from the origin, similar to the intercept in a linear equation, and is of crucial importance in the learning procedure. This process is repeated for each cell in subsequent layers until the output layer is populated.

To evaluate the value of any cell in the hidden or output layer in Figure 2.3(b), we use

$$a_j^{(l)} = \sigma \left[ \sum_{i=1}^{N} \left( a_i^{(l-1)} w_{i \rightarrow j}^{(l-1)} \right) + b_j^{(l)} \right],$$

where $a_j^{(l)}$ is the cell’s value with the subscripts and superscripts representing the cell and layer index, respectively; $\sigma$ is the non-linear activation function; and $N$ is the number of cells in layer $(l-1)$. Because weights connect two cells in successive activation function; and $N$ is the number of cells in layer $(l-1)$. Because weights connect two cells in successive layers, the subscript on $w_{i \rightarrow j}^{(l-1)}$ denotes the indices of the cells connected, while the superscript indicates their corresponding layer indices. Because a single bias value is added for each cell, $b_j^{(l)}$ has the same layer and cell indices as $a_j^{(l)}$. Only the weights and biases are updated
Figure 2.3 A sketch of a simple feed-forward neural network (FFNN). (a) The connections between the input layer and the first neuron in the hidden layer demonstrate how the value of that neuron is computed. (b) All connections from the input layer to the output layer.

during the training process. The other “hyper-parameters” are tuned for each application by trial and error based on the network performance during the testing/validation process in terms of matching the expected output.

The problem of finding the optimal weights and biases can be posed as an inverse (or optimization) problem. In each iteration, all inputs in the training data are injected into the input layer, and then an output is computed using the weights and biases of the current iteration. The predicted and expected outputs are compared using a similarity-measuring function, and the discrepancy between the two is used to update the network parameters through backpropagation [32].

To measure the difference between the predicted and expected outputs, we use the mean squared error (MSE) metric,

\[
E = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2,
\]

where \(y_i\) is the expected output for a given input in the training data, \(\hat{y}_i\) is the estimate predicted by the neural network (a function of the weights and biases of the neural network), and \(m\) is the number of model parameters. The MSE metric is commonly used in regression problems and is similar to the least-squares objective function commonly used in the data-domain FWI approach.
The training data set usually has a very large number of input-output pairs. If the above process were performed simultaneously on the entire training data set, the associated training cost can make this approach intractable. Therefore, we partition the set of training data into batches, with the batch size being a network hyper-parameter. We then update the network parameters more efficiently using the stochastic gradient descent method [42], where batches are inverted in parallel. This obviates the need to perform a simultaneous optimization process on a large number of training data. Passing the entire data set through the training algorithm completes an epoch. One then can test the network to examine the output predictions on data excluded from the training process.

2.4.1 Learning Generalization

In the field of machine learning, the term generalization usually refers to the ability of a neural network to predict the correct output of a set of input data not used in the network training. Most applications implicitly assume that the testing and training data have the same features and characteristics [43]. This represents an important test because low-loss networks that perform well on training data but poorly on data never before encountered are those that simply “memorized” the connection between the input and output training data and did not “learn” the underlying general connection.

Here, we further extend the term generalization to mean the ability of a neural network to predict outputs that exhibit totally different features and characteristics than the input data used in the training process. Using the four classes of velocity models described above, we investigate the ability of an ANN trained using data synthesized using one class of model to invert data synthesized using the same as well as other model classes. This allows us to examine questions such as how well does an ANN trained on a Type 2a model (i.e., squares without velocity gradients) estimate a Type 1b model (i.e., semi-geological with gradients)?

2.4.2 Convolutional Neural Networks

The ANN described above is an example of a fully connected feed-forward neural network (FFNN). This network simply consists of a number of layers (an input layer, one or more hidden layers, and an output layer), with each layer consisting of a number of neurons. Each neuron in a hidden layer is connected to every single neuron in the previous and following layers (hence “fully connected”).

A specialized and efficient ANN architecture consists of a sub-class of neural networks referred to as convolutional neural networks (CNNs). In contrast to FFNNs, the process of computing the weighted sum of all neurons in the previous layer is replaced by convolving all neurons in the previous layer with a convolutional operator (filter) of a fixed size and trainable weight coefficients. Similar to FFNNs, a
non-linear activation function is applied to the output of the convolution process plus the bias. The
convolutional layer is usually followed by a max-pooling layer, which is a filter of fixed size that slides
through the convolutional layer output and extracts the largest value. Hence, for a max-pooling filter of
size three, each three neurons in the output of the convolutional layer are replaced by a single neuron to
downsparse the output of the convolutional layer. For image recognition problems, using a max-pooling layer
leads to the benefit of incorporating positional invariance of a certain image feature [44]. We use it herein
to increase the efficiency of the training process.

To illustrate this process, Figure 2.4 shows the hidden layer in Figure 2.3 replaced by a 1-D
convolutional layer. Because the input and convolutional layers are fully connected (i.e., each neuron in one
layer is connected to each neuron in the other), the values at the first hidden layer (Figure 2.4b) are
computed using the process explained above for fully connected neural networks (i.e., equation 2.1).
However, because it is a convolutional layer, its output is computed differently. First, a filter of
pre-specified size (a CNN hyper-parameter) is applied to the input of the hidden layer (Figure 2.4c). In
this simple example, the filter has a size of three neurons and slides by a stride of one over the input to the
convolutional layer. At each convolution step, the filter weights are multiplied by the values of the
corresponding neurons and then summed to produce a single value that fills a single neuron in the
convolutional layer output. Moving the operator through all the neurons in the input to the convolutional
layer generates the layer output (Figure 2.4d). The max-pooling layer (Figure 2.4e) uses another filter of
size two sliding over the output of the convolutional layer and extracting the maximum value at each step
(Figure 2.4f). Because the output layer is fully connected to the max-pooling output, the values at the
output layer (Figure 2.4g) are computed using the procedure mentioned above for FFNNs (equation 2.1).
Convolutional layers can have more than one filter, each with distinct trainable weight coefficients. We use
this property of CNNs to invert frequency-domain seismic data by training a filter for each modeled
frequency component.

2.5 Experiments

2.5.1 ANN Architecture

For computational efficiency, we use a CNN with a single convolutional layer followed by a fully
connected layer. Deeper and more complex architectures provided only minimal improvements to the
predicted models accuracy at extra computational cost. The neural network used in our experiments
(Figure 2.5) begins with an input layer that receives the input seismic data as a 2-D array because the
input seismic data are sorted into frequency slices before injection into the neural network. Figure 2.6
illustrates how the input seismic data are sorted for a single frequency. The input layer is connected to a
Figure 2.4 A sketch of an ANN that has (a) an input layer, (b)-(d) a hidden convolutional layer, (e)-(f) a max-pooling layer, and (g) an output layer. (c) Shows how the convolution process is carried out using a sliding convolution operator, with the output of each convolution step placed in the corresponding cell in (d). Similarly, (e) shows a sliding max-pooling filter, with the output of each step placed in the corresponding cell in (f). The max-pooling layer is in turn connected to the output layer (g).
30-channel 1-D convolutional layer, where each channel is assigned to a single frequency in the input data (Figure 2.7). Hence, the convolutional layer has the same 30 channels as the input data and each channel has 150 filters (feature vectors) of size $2n_x$. We choose 1-D convolutional filters to treat the recorded data as a 1-D sequence of values (as opposed to a 2-D image). The filter size is specified to be $2n_x$ to concurrently handle the recorded real and imaginary data components corresponding to a single source. This can be seen clearly from the arrangement in Figure 2.6, where the $2n_x$ neurons for each channel carry the data from a single source. The convolutional layer is then connected to a max-pooling layer of size three, which is the maximum filter size that proved to be computationally efficient without losing resolution in the velocity model estimates.

![Figure 2.5 A sketch showing the architecture of the ANN used in all the experiments in this work.](image)
Figure 2.6 A sketch showing how seismic data for the $i$-th frequency ($0 \leq i < 30$) are sorted for input into a single channel of the convolutional layer. $n_x$ is the number of grid points in the $x$ direction, $N_s$ and $N_r$ denote the number of sources and receivers, respectively.

Figure 2.7 A sketch demonstrating how frequency-domain seismic data are injected into the input layer of the ANN. The convolutional layer has 30 channels, where each channel is assigned to a single frequency in the input seismic data.
After the max-pooling layer, we have a fully connected layer with a size of $4n_z n_x$, which in turn is connected to an output layer of a size equal to the desired number of output grid points (i.e., velocity models of $n_z \times n_x$ grid points). We used the rectified linear unit (ReLU) activation function for the convolutional and fully connected layers. No activation function was used for the output layer to allow it to have any value from the permissible range of output velocities. Due to the relatively small number of network parameters (i.e., weights and biases), we did not include any regularization term (e.g., dropout layers). All hyper-parameters were chosen by trial and error. Different ranges were tested and the values that produced the best compromise between accuracy and efficiency were selected.

### 2.5.2 Training

We trained four CNNs using the architecture described above using the four types of training data (i.e., Types 1a/1b and 2a/2b). A total of 10,000 input-output pairs for each data type were used to train the corresponding neural network. After passing each batch of training data through the network, we computed the misfit between predicted and true outputs, and used it to update the trainable network parameters (i.e., weights and biases). We computed the misfit using the mean squared error (MSE) loss function (equation 2.2). Due to the large size of the training data, we performed parameter updating using stochastic gradient descent, where the training data were split into a series of smaller individually processed batches (400 input-output pairs per batch).

To assess the network learning, we plot the training and the Type 1b validation dataset loss function values at each epoch (i.e., the learning curve) for each network. Because we are interested in assessing how well each network reconstructs layered velocity models, we use the same Type 1b validation dataset when training all four networks. This means that the validation loss tests network generalization in so far as it reflects how well each network performs when tested on data not used during the training process. Decreasing training and validation curves with increasing epochs indicates that the network is learning. However, a decreasing training curve with an increasing validation curve suggests that the network is overfitting parameters to the training data.

Figure 2.8a-Figure 2.8d show the learning curves for the four networks. For Type 1a (Figure 2.8a) and Type 1b (Figure 2.8b), the divergence between the training and validation curves occurs around 450 epochs; however, for Type 2a (Figure 2.8c) and Type 2b (Figure 2.8d), the divergence occurs earlier at around 150 epochs. Beyond these divergence points, the recovered validation models appear to be of higher resolution; however, this is due to the inclusion of the desired short-wavelength model features as well as non-physical short-wavelength noise. We inspected the recovered validation models for each network and determined the stopping epoch based on a compromise between resolution improvements and...
high-frequency noise. After training, we tested all four networks on the 10,000 Type 1b testing data, which is a useful test because some models have velocity gradients while others do not. Accordingly, we can assess the influence of velocity gradients in the training or testing data on the predicted models.

![Learning curves for the four CNNs trained using (a) Type 1a data, (b) Type 1b data, (c) Type 2a data, and (d) Type 2b data.](image)

**2.5.3 Experiment 1: CNN Trained on Type 1a/1b Velocity Models**

Figure 2.9 shows (typical) results of testing the neural network that has velocity gradient in its training data (Type 1b neural network). Figure 2.9a presents the testing (true) velocity model, Figure 2.9b shows the model predicted using Type 1b neural network, and Figure 2.9c presents a “pseudo-well” cross-section through the model extracted at the dashed line. Overall, this example shows a good performance of the neural network in predicting the layer shape and velocity structure. The second testing model (Figure 2.9d) has more complex velocity structure with layers of different dips, followed by a thick dipping layer with a linear velocity gradient, and then a strong velocity inversion at the bottom. The network again does a good job in recovering the shapes and velocities of all layers (Figure 2.9e and Figure 2.9f). The only missing features in the recovered model are the sharp boundaries for the shallow layers, which is likely attributed to these layers being too thin to be detected by the 3-32 Hz frequencies used in the experiment. Similar behavior occurs in Figure 2.9g-Figure 2.9i where the network recovers a smoothed version of the third testing model, which could be caused by the minimal intra-layer velocity variations compared to most models used in network training. Overall, Type 1b network performed well for estimating a range of layer thicknesses and dips as well as for recovering velocity inversions. The observed smoothing may be
attributed to the nature of the training data, with Type 1b training effectively enforcing a smooth transition between layers.

Figure 2.9 Testing results of the neural network trained on Type 1b models and tested on Type 1b models having the same structure. (a), (d), and (g) are the testing velocity models. (b), (e) and (h) are the corresponding predicted models. (c), (f) and (i) show the cross-section through each of the corresponding models at the dashed line (pseudo-well locations). The same color bar is used for all velocity models.

To assess the influence of training model velocity gradients on the performance of the network, a second neural network was trained using only Type 1a models (i.e., no velocity gradients) and tested on the same models we used to test Type 1b network (Figure 2.9). Figure 2.10 presents the test results. The velocity model recovered in Figure 2.10b shows a similar behavior to that shown in Figure 2.9b in the shallow
section, even estimating the velocity gradient with high accuracy. However, it poorly recovers the velocity structure of the bottom layer. This behavior is representative of using a Type 1a trained network to estimate Type 1b testing models. This is also observed in Figure 2.10e, where the velocity gradient causes the network to recover an incorrect velocity for the bottom layer. In general, whether the velocity increases or decreases below the velocity gradient, the bottom layer is almost always assigned a velocity that is close to the value in the overlying layer. However, this is not always because the model in Figure 2.10h recovers the structure and velocity variations at the bottom right corner of the model slightly better than Figure 2.9h.

While these examples present a qualitative analysis of how Type 1a/1b networks perform when tested on certain geological structures, they do not provide a statistical analysis on how each network performs when tested on a large number of geological models with widely varying complexity. To do so, we use the 10,000 Type 1b models reserved for testing (from which the above examples were extracted) to test the two trained networks discussed in this section. Figure 2.11 present the normalized root-mean square (rms) error for the two trained networks. The statistical results confirm the conclusions of our qualitative analysis. Figure 2.11b shows that models predicted using Type 1b network generally reconstructs velocity models that better fit the true models, followed closely by models predicted using Type 1a network (Figure 2.11a). This observation is supported by the value of the median error for Type 1b networks (0.130) being lower than that of Type 1a networks (0.170).

2.5.4 Experiment 2: CNN Trained on Type 2a/2b Velocity Models

The motivation behind the second experiment is to evaluate whether neural networks can learn the seismic inversion problem using purely geometric velocity models, or whether we need to explicitly use layering in our training data. Figure 2.12 presents the testing results for the Type 2b trained network. It is intriguing to see how a neural network that has not been trained on coherent geologic-like structures can recover a good approximation of the shape of some semi-geological layers. A key observation here is that the network learned what square sizes are needed to reconstruct each geologic structure depending on its size and shape. Figure 2.12b shows the network using small squares to reconstruct the curved interface between the second and third layers, while in Figure 2.12e and Figure 2.12h the same network makes use of mainly larger squares to recover broader structures. Due to this behavior, the CNN performed well for the shallow layer in Figure 2.12h because its near-rectangular shape. However, in Figure 2.12e the use of large squares poorly reconstructed the curvature of the interface between the second and third layers.

Fig/Type2bExperimentwidth=1Testing results of the neural network trained on Type 2b models and tested on Type 1b models. (a), (d), and (g) are the testing velocity models. (b), (e) and (h) are the
Figure 2.10 Testing results of the neural network trained on Type 1a models and tested on Type 1b models. (a), (d), and (g) are the testing velocity models. (b), (e) and (h) are the corresponding predicted models. (c), (f) and (i) show the cross-section through each of the corresponding models at the dashed line (pseudo-well locations). The same color bar is used for all velocity models.
Figure 2.11 Histograms showing the normalized rms error for the networks trained on (a) Type 1a, (b) Type 1b, (c) Type 2a, and (d) Type 2b data and tested on Type 1b data.

corresponding predicted models. (c), (f), (i) show the cross-section through each of the corresponding models at the dashed line (pseudo-well locations). The same color bar is used for all velocity models.

A second important observation is that the accuracy of the recovered models decreases with depth. Shallow layers are mostly recovered with a moderate-to-high accuracy, but the quality of the predicted models demonstrably deteriorates with depth. This is a common feature in most of the recovered models in this experiment. In rare cases, large non-physical structures appear at the model bottom (e.g., Figure 2.13h). While this might mean that this type of purely geometric training data is better suited for relatively near-surface inversion applications, this observation is more likely due to limited aperture or illumination effects. A further limitation of the Type 2a/2b models is that the network fails to maintain an accurate intra-layer velocity, especially at depth. Although it well approximates that layer shape and velocity in the vicinity of the interface, the velocity erroneously varies significantly within the layer.

Figure 2.13 presents the Type 2a testing results and shows only moderate differences compared to the examples in Figure 2.12. Figure 2.13 also illustrates that the neural network accurately recovered velocity gradients for the shallow layer without needing to be explicitly trained on models with velocity gradients. This can be explained by the fact that numerous small, vertically stacked squares with minor and smooth velocity increases or decreases can have the same effect on seismic data as larger squares with explicit velocity gradients. In addition, the three cross-sections in Figure 2.13 are less noisy than the corresponding cross-sections in Figure 2.12. This could indicate that much of the apparent heterogeneity in the recovered
Figure 2.12 Testing results of the neural network trained on Type 2b models and tested on Type 1b models. (a), (d), and (g) are the testing velocity models. (b), (e) and (h) are the corresponding predicted models. (c), (f), (i) show the cross-section through each of the corresponding models at the dashed line (pseudo-well locations). The same color bar is used for all velocity models.
models in Figure 2.12 is due to the network training on models exhibiting frequent velocity changes throughout, including inter- and intra-square velocity variations.

Figure 2.13 Testing results of the neural network trained on Type 2a models and tested on Type 1b models. (a), (d), and (g) are the testing velocity models. (b), (e) and (h) are the corresponding predicted models. (c), (f) and (i) show the cross-section through each of the corresponding models at the dashed line (pseudo-well locations). The same color bar is used for all velocity models.

Similar to the previous section, a statistical analysis was performed using Type 2a/2b networks (Figure 2.11c and Figure 2.11d). The rms error distribution is consistent with our observations in the above qualitative analysis. As expected, Type 2b networks have a larger median rms error than Type 1a/1b networks (0.235), but still smaller than for to Type 2a networks (0.241). Although models predicted
using Types 2a/2b appear to be less accurate than Type 1a/1b networks, their misfit values are still reasonably small, given that they were constructed using squares only in the training data.

Despite the shortcomings of Type 2a/2b-trained networks, we believe their overall performance offers insight on how the seismic velocity model building problem is treated by ANNs. The ability to combine squares of different sizes to construct geologic-like structures similar to the true ones has implications for ANN-based methods. In particular, that they can estimate velocity models based on the “degree of completeness” of the training process without relying on the relative accuracy of a single starting velocity model like in FWI.

2.5.5 Scale Analysis of Type 2a/2b models

The Type 2a/2b experiments described above incorporated a wide range of squares sizes in the training data. While these CNNs recovered testing models with moderate accuracy, an analysis of the significance of different square-size ranges in the training data would provide an insight on how the CNN is automatically learning the appropriate usage of each square-size range. To assess the significance of small-sized squares in the training data, we generated three Type 2b training data sets with square sizes ranging between 5 – 25, 10 – 25, and 15 – 25 grid points (i.e., fixing the maximum square size while altering the minimum). Three CNNs were trained using the three newly generated training data sets and tested on two Type 1a/1b models (Figure 2.14a and Figure 2.14g); one has a curved interface that can only be recovered using small squares, and the other consists of large, almost-flat layers that can be recovered using larger squares.

Figure 2.14 compares the models predicted using the original Type 2b network (with squares ranging between 1-25 grid points) and each of the three newly trained networks. While the recovered models are similar, there are subtle differences between the predicted models shown in Figure 2.14. Training with smaller squares sizes allows for greater freedom in constructing curved interfaces. However, this also causes increasing intra-layer velocity variations with decreasing smallest square size used in the training data. This is because recovering a uniform layer velocity requires that all squares inside the layer be assigned the same true velocity value. As the size of these squares becomes smaller, their number increases, and assigning the true velocity to a large number of small squares appears to be a challenging task for CNNs trained on Type 2a/2b models. Figure 2.14e shows another notable result where the smallest square in the training data is too large to recover the curved interface, causing bottom layer of the recovered model to diverge from the true one.

To assess the significance of large squares in the training data, we repeated the previous experiment but using square sizes ranging between 1 – 5, 1 – 10, and 1 – 15 grid points. Figure 2.15 presents the predicted models for each square size range using the same testing models as previously. Changing the lower limit of
square sizes in the training data has a more pronounced effect on the predicted models than changing the upper limit. Figure 2.15b and Figure 2.15h demonstrate that intra-layer velocity variations in Type 2a/2b models are mainly due to the inability of CNNs to recover the correct velocity for a large number of small squares. However, as we increase the upper limit of square sizes in the training data, we observe improved continuity of velocity values inside the layers. Therefore, based on these experiments, we infer that CNNs trained on Type 2a/2b models use small squares to recover non-flat model features, and large squares to reconstruct longer-wavelength velocity model features.

2.5.6 Experiment 3: The Marmousi Model

To further assess how training model structure influences neural network performance in solving the seismic inversion problem, we test the four network types trained in the previous sections on an “independent” velocity model not used in the training process of either network. For this experiment, we chose 600 of \( n_x \times n_z = 50 \times 50 \) grid-point subsets from different parts of the Marmousi model. We used these velocity models to synthesize seismic data using the same modeling parameters described above (i.e., source and receiver locations, frequency range, and source wavelet). The synthesized data were then input to the four trained networks.

Figure 2.16-Figure 2.18 show the results for three selected subsets. Figure 2.16b and Figure 2.16c show a comparison of predictions made by Type 1b and 1a networks (with and without velocity gradients). We deem the results obtained using Type 1b network to be superior because they better maintain the true velocity throughout all the layers, whereas the Type 1a network erroneously recovers more significant intra-layer velocity variations. The same behavior is observed for Type 2b and 2a network results shown in Figure 2.16e and Figure 2.16f. The model predicted using the Type 2b network recovers the location and shape of the shallow interface with a much higher accuracy than the Type 2a network and better maintains the true velocity throughout each layer. Overall, the Type 1b network produced the most accurate velocity model. Although the true velocity model has no explicit velocity gradient in any of its layers, it contains a number of thinly stacked layers with slowly varying velocities. The resulting implicit velocity gradients may be the reason why Type 1b and 2b networks produced superior results.

We repeated the above experiment for a slightly more complex subset of the Marmousi model (Figure 2.17). The results obtained from the Type 1b/1a networks are similar (Figure 2.17b and Figure 2.17c) in terms of the recovered shape and velocity of the shallow dipping layers. In Figure 2.17e, though, the performance of the Type 2b network is intriguing because it recovered the correct dips and velocity values of the shallow section. Similar to our observations in Figure 2.12b and Figure 2.13b for curved interfaces, this test shows how the CNN also learned to use small squares to reconstruct the shape.
Figure 2.14 Testing results of the neural networks trained on Type 2b models with square sizes ranging between (b) and (h) 1−25, (c) and (i) 5−25, (d) and (j) 10−25, and (e) and (k) 15−25 grid points. (f) and (l) present the cross-sections for the recovered models of the first testing model (a). (f) and (l) present the cross-sections at the dashed lines (pseudo-well locations) for the recovered models of the first and second testing models, respectively. The same color bar is used for all velocity models.
Figure 2.15 Testing results of the neural networks trained on Type 2b models with square sizes ranging between (b) and (h) $1 - 5$, (c) and (i) $1 - 10$, (d) and (j) $1 - 15$, and (e) and (k) $1 - 25$ grid points. (f) and (l) present the cross-sections at the dashed lines (pseudo-well locations) for the recovered models of the first and second testing models, respectively. The same color bar is used for all velocity models.
of dipping layers. The Type 2a network performed equally well in recovering the shallow section of the model; however, it failed to recover the correct velocity of the deeper section, which is in agreement with the previous observations for Type 2a/2b networks.

Finally, Figure 2.18 shows the testing results on a more structurally complex subset of the Marmousi model. None of the four neural networks recover the true velocity model to the same level of accuracy as observed in Figure 2.16 and Figure 2.17. Type 1b/1a networks (Figure 2.18b and Figure 2.18c) perform better in the section above the fault than below it. This suggests that the CNNs were “confused” by the faulted layering, since this type of geologic feature was not incorporated in the training data sets. Conversely, the general nature of Type 2b training models means that this network is less biased toward specific geologic structures. In previous experiments, this network recovered both curved and dipping interfaces by making use of training on square of different sizes. This training generality is also reflected in Figure 2.17f where the Type 2b network performed slightly better than the Type 1b network in maintaining the true velocity of the shallow layers and recovering the velocity structure below the fault. As established in previous experiments, Type 1a and 2a networks recover models exhibiting lower velocity variation, but with lower global accuracy.

As with Type 1b testing models, different subsets of the Marmousi are composed of different geological structures of different complexity. To better understand how each network performs with a large number of such “independent” models, we test the four networks using 600 Marmousi model subsets and perform a statistical analysis similar to that performed with Type 1b testing data described above. We compute the rms error between the true and each recovered model (Figure 2.19). Figure 2.19 displays the same behavior as Figure 2.11. Type 1a and 1b trained networks (Figure 2.19a and Figure 2.19b) have median rms errors of 0.050 and 0.047, respectively. Models predicted using Type 2a and 2b trained networks have a similar accuracy (Figure 2.19c and Figure 2.19d) with median rms errors of 0.111 and 0.110, respectively.

### 2.6 Analysis of Recovered Models

To further assess the velocity model building capability of the trained networks, we investigate how well seismic data synthesized using the recovered models fit the input data. To perform this analysis, we simulated frequency-domain data using the ANN-recovered velocity models and compared the true and simulated seismograms for each frequency. Figure 2.20 shows an example of the data-fitting comparison for the models recovered in the Type 1a/1b testing experiment (i.e., Figure 2.9 and Figure 2.10) for the magnitude and phase components at three frequencies (3 Hz, 18 Hz, and 32 Hz). The true and simulated 3 Hz data are almost identical; however, the two begin to diverge as the frequency increases until they reach their maximum misfit at 32 Hz. This behavior is consistent with the fact that the recovered velocity
Figure 2.16 The first Marmousi model subset used to test the four trained networks. (a) Marmousi model subset. Prediction results for (b) Type 1b, (c) Type 1a, (e) Type 2b, and (f) Type 2a training. (d) and (g) present the cross-sections corresponding to Types 1a/1b and 2a/2b, respectively. The same color bar is used for all velocity models.
Figure 2.17 The second Marmousi model subset used to test the four trained networks. (a) Marmousi model subset. Prediction results for (b) Type 1b, (c) Type 1a, (e) Type 2b, and (f) Type 2a training. (d) and (g) present the cross-sections corresponding to Types 1a/1b and 2a/2b, respectively. The same color bar is used for all velocity models.
Figure 2.18 The third Marmousi model subset used to test the four trained networks. (a) Marmousi model subset. Prediction results for (b) Type 1b, (c) Type 1a, (e) Type 2b, and (f) Type 2a training. (d) and (g) present the cross-sections corresponding to Types 1a/1b and 2a/2b, respectively. The same color bar is used for all velocity models.
Figure 2.19 Histograms showing the normalized rms error for the networks trained on (a) Type 1a, (b) Type 1b, (c) Type 2a, and (d) Type 2b data and tested on 600 subsets of the Marmousi model.

models are mostly smooth and, in most cases, do not contain sharp discontinuities.

Figure 2.21 shows the data-fitting test corresponding to the velocity models recovered using Type 2a/2b networks (i.e., Figure 2.12 and Figure 2.13). Interestingly, the accuracy of the simulated data does not appear to be greatly affected by the pixellated structure of the recovered models even at the highest frequency in the data. Therefore, though the structure of the models recovered using Type 2a/2b networks appears to have an imprint of the square structure, the seismic data appear insensitive to this within the range of frequencies and acquisition geometry used in this experiment.

To evaluate the data-fitting capability of the trained networks statistically, we use the same 10,000 Type 1b testing data used in the model-fitting statistical analysis. We simulated seismic data for the 10,000 models predicted by each trained network and computed the rms error between true and synthesized seismic data. Figure 2.22(a), Figure 2.22(d), Figure 2.22(g), and Figure 2.22(j) presents the 3 Hz rms differences where the median rms misfits for Type 1a and 1b networks are 0.039 and 0.032, respectively. Conversely, there is a broader distribution of the data misfit for Type 2a and 2b networks with the median rms error being 0.098 and 0.109. This is likely related to the fact that the model misfit for Type 2a/2b networks shows a similar distribution (Figure 2.11). The 18 Hz results (Figure 2.22(b), Figure 2.22(e), Figure 2.22(h), and Figure 2.22(k)) show a wider range of rms error with an overall shift to higher values for all four networks. This trend is even more apparent at 32 Hz (Figure 2.22(c), Figure 2.22(f), Figure 2.22(i), and Figure 2.22(l)), where the rms error is greatest. The median rms misfit
increases to 0.114 for Type 1a, 0.099 for Type 1b, 0.167 for Type 2a and 0.163 for Type 2b networks.

Overall, the model and data misfit analysis provides clear evidence that the models recovered using all four networks are of moderate to high accuracy; the Type 1b results are the most accurate and followed closely by Type 1a, and then the Type 2a/2b networks.

![Figure 2.20](image)

Figure 2.20 Comparison of the magnitude and phase components of the data simulated using Type 1b/1a models. (a) True model. (b) Type 1b result. (c) Magnitude and phase comparisons for data modeled using result in (b) for 3 Hz, 18 Hz and 33 Hz. (d) Type 1a results. (e) Type 1a result. (f) Magnitude and phase comparisons for data modeled using result in (d) for 3 Hz, 18 Hz and 33 Hz.

2.7 Discussion

We chose a small model size in this work mainly for computational efficiency purposes. A large number of ANN training experiments needed to be performed concurrently to optimize the network hyper-parameters (e.g., ANN architecture, number of layers, number of cells in each layer, the number of convolutional filters and their sizes). In principle, extending the obtained results to larger models with dimensions closer to realistic field data settings should be straightforward because the hyper-parameters implemented here are a function of model dimensions (i.e., the CNN 1-D filter size should be the same as the total number of recorded seismic data points per source). However, the bottleneck for ANN training is the GPU memory requirement, which increases linearly with either the depth (number of layers), the width
Figure 2.21 Comparison of the magnitude and phase components of the data simulated using Type 2b/2a models. (a) True model. (b) Type 2b result. (c) Magnitude and phase comparisons for data modeled using result in (b) for 3 Hz, 18 Hz and 33 Hz. (d) Type 2a results. (e) Type 2a result. (e) Magnitude and phase comparisons for data modeled using result in (d) for 3 Hz, 18 Hz and 33 Hz.
Figure 2.22 Histograms showing the normalized data-fitting rms error. The left, center and right columns are for 3 Hz, 18 Hz and 32 Hz data, respectively. (a)-(c) present results for Type 1a data, (d)-(f) for Type 1b data, (g)-(i) for Type 2a data, and (j)-(l) for Type 2b data.
(number of neurons per layer), as well as crossline dimension for 3-D velocity inversion problems.

The different square sizes in Type 2a/2b models is inspired by the multi-scale approach commonly used in FWI. Type 2a/2b experiments provide insight as to whether an ANN can automatically learn the multi-scale approach without needing to explicitly incorporate it in the training process. Such an approach could obviate the expensive process of incrementally adding frequencies at later iterations like in FWI velocity model building. Based on these results, we speculate that results may be improved by explicitly incorporating the multi-scale approach in the training process. That is, training could start with the lowest frequency available in the data along with the largest square size, and then sequentially increasing the input data frequencies while decreasing the square size.

Because this is a relatively new methodology, insufficient research work has been reported on analyzing how changing the number of layers and the number of neurons in each layer (or the number convolutional filters and the size of each filter) would alter the inversion results. Similarly, experimentation on how changing each hyperparameter affects the inversion results is limited. Therefore, building an optimized ANN and specifying its hyper-parameters remains a daunting task. We believe that these challenges and associated computational costs will be overcome as ANN-based seismic inversion approaches the scientific maturity of FWI. Once that level is reached, the iterative ANN training process could become more efficient than FWI because, in addition to using GPUs highly optimized for neural network training, the training process is completely automated unlike FWI iterations that often require human interaction. In addition, an ANN trained on seismic data from a certain geological basin could be used to invert seismic data from different parts of the basin or acquired at different times. This would likely be more efficient than FWI approaches where the full iterative process is repeated for each newly acquired seismic data set.

We speculate that combining the two model classes presented here (i.e., “geological” and “geometric”) may be necessary to achieve an optimal overall training generalization. Specifically, a network that uses semi-geological models, but is free to fall back on purely geometrical shapes to reconstruct features not included in the semi-geological training data set. The general nature of this combination could make such a network more attractive as it may forestall the need to explicitly model a complete “dictionary” of all geological features likely to be encountered in the field-data velocity model building process.

2.8 Conclusions

We demonstrate the significant potential in harnessing the power of ANNs to solve the seismic velocity inversion problem. No background velocity models were provided as an input to the CNNs, and all models were predicted based on minimal information after network training. This property of ANNs could be very significant for mitigating the often severe non-linearity problem of conventional seismic velocity inversion
methods. Because neural networks are data-driven tools, though, their performance is mainly controlled by the data used in the training process. While we do not need to provide an ANN with all possible velocity models for it to recover meaningful seismic velocity inversion results, we still must provide it with a representative and sufficient sample of models during the training process. Our analysis provides insight on what features or structures need to be included in the velocity model training data and how the inclusion or exclusion of such structures can affect overall network performance.

We generated four sets of training velocity models: two “semi-geological” and two purely geometric, both with and without velocity gradients. Each testing velocity model in the previous section revealed the advantages and limitations of a particular choice of training data. Based on these results, we generally prefer using semi-geological training data because the predicted model results usually were more continuous and exhibited less unwarranted short-wavelength velocity variation. Scenarios where the testing models included a velocity gradient or a sequence of thin layers of uniform or slowly varying velocity required semi-geological models with velocity gradients to recover a sufficiently accurate estimate. However, if the testing models consisted of blocky layers, then the semi-geological models without velocity gradients are preferred because the inclusion of training velocity model gradients introduces unwanted model smoothing effects.
CHAPTER 3
SEISMIC VELOCITY MODEL BUILDING USING RECURRENT NEURAL NETWORKS: A FREQUENCY-STEPPING APPROACH

3.1 Abstract

Data-driven artificial neural networks (ANNs) demonstrably offer numerous 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; 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 velocity model building approach 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 or chosen, while the corresponding output sequences are frequency-dependent smoothed velocity models. We compare models recovered using the trained RNN to the true models qualitatively and quantitatively. The normalized root-mean-square (NRMS) misfit between the true and predicted models has a mean of 6%, which confirms that the network recovers highly accurate models.

3.2 Introduction

Seismic waves are used as a remote sensing tool to investigate the earth’s interior. Seismic data are most often acquired by placing a seismic source and a number of receivers on the earth surface, exciting the source, and recording transmitted as well as reflected wavefield energy. By repeating this experiment for a number of sources at different locations, one can obtain good illumination subsurface targets for the purpose of structural imaging and material property inversion.

A major challenge in these seismic exploration activities is building subsurface velocity models that minimize the misfit between observed and simulated seismic data. Over the past decades, this challenge has been tackled using mainly deterministic methods. 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 minimizes the misfit between observed and modeled data [7, 8]. Because the governing wave equation is non-linear with respect to medium parameters, the FWI process is itself highly non-linear, which leads to non-convex objective functions with numerous local minima that frequently forestall the convergence of iterative optimization approaches toward the globally minimum solution [9]. Such
challenges are exacerbated by the oscillatory nature of seismic data, which consequently demands using starting models that simulate 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. FWI is also computationally expensive because each iteration usually requires (tens of) thousands of 3D seismic wave simulations.

To mitigate the non-linearity issues, numerous FWI approaches follow a multi-scale strategy 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 [11]. Because lower frequencies are less prone 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 band incorporating higher frequency data. This iterative approach then continues until inversion of the final frequency band is completed.

The resurgence of machine learning (ML) techniques in recent years, largely driven by advances in GPU computing, is introducing new paradigms for automated velocity model building that do not inherently suffer from the aforementioned challenges. 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 by the trained network can be made at the cost of matrix-vector multiplications. The challenges of seismic inversion have been addressed by a number of authors. Some authors (e.g., [45] and [23]) succeed in using ANNs to extrapolate low frequencies in the seismic data, thus improving the convergence of FWI. However, the majority of published ML-based work aims to use ANNs to perform the inversion themselves. [39] and [27] use convolutional neural networks (CNNs) to invert common mid-point (CMP) semblance cubes, with [27] using transfer learning to invert field data. [46] uses a fully convolutional network (FCN) to construct velocity models with salt bodies. [47] uses an autoencoder architecture to map seismic data to their corresponding velocity model. [48] builds an ANN that constructs velocity models by accounting for the spatial correspondence between seismic data and their corresponding velocity models. [28, 49, 50] use generative adversarial networks (GANs) [51] to solve the seismic velocity model building problem.

Motivated by the multi-scale approach of FWI, 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 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. Previous attempts to use ANNs for seismic velocity model building were all performed in the time domain. Therefore, they require the entire seismic data to be simultaneously fed into the network. However, the proposed method only requires a small number of discrete frequencies that are successively fed into the network. Furthermore, at each frequency step, only the data corresponding to that frequency need to be loaded into memory.

We begin by introducing RNNs and their different types including the benefits of using long short-term memory (LSTM) 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.

3.3 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 have a constant velocity. Figure 3.1 shows 16 representative 2 km × 2 km P-wave velocity models comprised of layers with a range of dip angles and thicknesses. Some layers are curved in the shape of synclines; others have the form of anticlines.

We then use the developed velocity models for numerical experimentation. A total of 17,000 input-output pairs are generated, of which 10,000 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 simulate 2D seismic data using a GPU-based constant-density acoustic wave-equation solver based on a $O(\Delta x^8, \Delta t^2)$ finite-difference time-domain (FDTD) numerical approach with an active free surface assuming a 15 Hz Ricker wavelet source. We then apply a temporal FFT to transform the data set and extract data at nine frequencies $f$ 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.
Figure 3.1 16 representative 2.0 km × 2.0 km synthetic velocity models generated for network training.
3.4 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. In addition to 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 CNNs and 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.

Another prominent feature of RNNs is that they have a “hidden state” vector that acts as the network’s memory and helps it “recall” what it has learned between the first and final steps. There are different types of RNNs depending on the number of network inputs and outputs, the most basic of which is the one-to-one RNN that yields a single output for a single input. Figure 3.2a illustrates such a network where the curved arrow indicate the recurrence feature, which simply means that a certain hidden layer has an associated hidden-state vector as shown in Figure 3.2b. 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.

One example of using sequentially decomposed data can be found in machine translation applications, which show that feeding the input text word by word to the ANN produces superior results than when feeding the entire text at once [52]. For a fully connected network, this is because each input word in a sentence is translated independently without examining its context. However, using a one-to-one RNN with a hidden-state vector containing information about the previous network inputs can help convey the
context of the input word in the sentence and lead to better translation results.

A second application where RNNs are frequently used is to solve the problem of captioning a video or a sequence of images. Figure 3.3 shows a comparison between feed-forward and recurrent neural networks. In a feed-forward network (Figure 3.3a), each image is input into the network independently. The network will only recognize that the main object in each image is an airplane; it has no way of recalling the images observed before the current one and thus loses out on an opportunity to discover the context of the image sequence. However, for the RNN in Figure 3.3b, each hidden layer has a hidden-state vector that acts as its memory. Therefore, when the last image is input to the network (i.e., the scenario depicted in Figure 3.3b), the network already has information about all previous images through information stored in the hidden-state vector. Therefore, hidden-state vectors can provide the network with contextual information required to recognize the sequence of images as an airplane taking off. These properties of RNNs prompted their use in a number of geophysical applications. For example, [53] and [54] use RNNs with common mid-point (CMP) gathers as the network input to recover 1D velocity profiles that respectively represent stacking and interval velocity.

In addition to one-to-one RNNs there is the many-to-many (or sequence-to-sequence) RNN (Figure 3.2c) that we use here to produce optimal results. In the sequence-to-sequence RNN, the vectors \( h_1 \) to \( h_n \) represent the network hidden state vectors, \( x_1 \) to \( x_n \) the inputs vectors, and \( y_1 \) to \( y_n \) the corresponding output vectors. The training process starts by injecting the first input \( x_1 \) into the network, along with its corresponding output \( y_1 \). Vector \( h_0 \) is updated based on what the network has learned from the connection between \( x_1 \) and \( y_1 \) to produce \( h_1 \). The process is repeated for the remaining input-output pairs. The final output \( y_n \) is computed using the continually updated state vector \( h_n \) that, in principle, carries the cumulative network learning along with the final input \( x_n \).

The network structure in Figure 3.2c 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 \( x_1 \) vector, followed by sequentially higher frequencies until the highest frequency data are injected as \( x_n \). The output vector at each step, \( y_1 \) to \( y_n \), ideally contains the model wavenumber components corresponding to the highest frequency used to that point.

Because there is nothing in the sequential RNN processing that forces models developed from low-frequency data to contain only low-wavenumber information, we need to introduce an additional operator to effectively restrict model wavenumber content. (Note that this process is analogous to “conditioning the gradient” in FWI.) Because such filtering operators are not straightforward to construct for heterogeneous models, we globally apply a 2D Gaussian convolutional filter with an adaptable radius (i.e., \( \sigma_s \)) to smooth velocity models to a degree commensurate with the maximum frequency used to
Figure 3.3 In a feed-forward neural network (a) each image is fed independently, leading the network to recognize the object in each image as an airplane. However, in a recurrent neural network (b) the hidden-state vector acts as the network memory so that as we inject the last image in the sequence, information about the previous images had been stored in the hidden-state vector. This leads the network to recognize the movement of the object in the images as an airplane taking off.
construct it. Figure 3.4 illustrates the application of the 2D Gaussian convolutional filter on an example velocity model for $\sigma_s$ values of 18, 10 and 2. The choice of $\sigma_s$ clearly controls the wavenumber content of the resulting smoothed model.

Figure 3.4 Three examples of the 2D Gaussian smoothing operator used in this work. When the Gaussian smoothing radius ($\sigma_s$) is 18, the smoothing operator is large and the smoothed velocity model only has the general trend of the true velocity model. As we reduce $\sigma_s$, the smoothing operator becomes smaller and more velocity structures are preserved in the smoothed model. Large filters are used for smaller frequencies, while smaller filters are used for higher frequencies.

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).

3.4.1 Long short-term memory RNNs

A common drawback with standard RNNs is the vanishing or exploding gradient problem [55]. To address this issue, a particular class of RNNs called long short-term memory (LSTM) has been developed [56]. LSTMs can hold information for a longer duration by introducing a gating mechanism at each step to determine the relevant information to be carried forward to the next step in the sequence and to discard irrelevant information. In addition to the hidden-state vector that serves as the short-term network memory, an LSTM layer introduces a cell-state vector that serves as the network’s long-term memory.

Figure 3.5 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.

Figure 3.5a and Figure 3.5b illustrate how information flows inside RNN and LSTM cells, respectively. An RNN cell (Figure 3.5a) has one layer and two inputs: the hidden state vector coming from the previous cell ($h_{t-1}$) and the network input at the current step ($x_t$). These two input vectors are passed to a layer that has two set of weights and biases, one for $h_{t-1}$ and the other for $x_t$. Each vector is multiplied by its corresponding weight matrix and added to its corresponding bias vector. The activation function is applied to the sum of the two resulting vectors to produce the updated hidden state vector $h_t$.

In contrast with the RNN cell, an LSTM cell has four trainable layers. In Figure 3.5b each light green box represents a layer, with its own set of weights and biases learned during training, and a predefined
activation function. The four layers constituting an LSTM cell are the forget, input, candidate-cell-state, and output layers \((f_t, i_t, \tilde{C}_t, o_t)\). 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. The activation function, commonly denoted by \(\sigma\), 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. Each layer in an LSTM 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. The LSTM cell has two outputs, the updated cell-state vector \(c_t\) and the updated hidden state vector \(h_t\), which are passed to the next LSTM cell in the same layer. For multi-layer RNNs, the updated hidden state vector (of both simple RNN cells or LSTM cells), is passed as an input to the corresponding cell in the next layer.

To simplify the operations of the LSTM cell shown in Figure 3.5b, 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. 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 information that the network finds to be irrelevant, 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 updated cell-state vector \(c_t\) and multiply the resulting vector with \(o_t\). Therefore, the equation for computing the updated hidden state vector is \(h_t = \text{ReLU}(c_t) \times o_t\).

Looking at Figure 3.5 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.

### 3.4.2 LSTM-RNN Velocity Inversion

Figure 3.6 shows the network architecture used in our experiment (starting at the bottom and moving upward). Each gray square represents an LSTM cell, which are combined in a sequence to form LSTM layer. In the context of the present experiment, the sequence length represents the number of frequency steps used 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. To illustrate how information flows through the network in a forward pass (i.e., computing velocities from seismic data), Figure 3.7 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\textsubscript{i0} and c\textsubscript{i0} for i = 1, 2, 3) are all initialized to zero. A forward run of the network in Figure 3.7 begins by injecting the 1.1 Hz seismic data for all sources as the x\textsubscript{1} vector. Using the weight matrices of LSTM\textsubscript{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\textsubscript{11} and cell-state c\textsubscript{11} vectors.

During the next stage the network proceeds both vertically (to the next layer) and horizontally (to the next frequency). We will first consider the vertical propagation of information. The LSTM\textsubscript{12} cell takes h\textsubscript{11} as its input vector and null vectors h\textsubscript{20} and c\textsubscript{20} and uses the layer weights to compute h\textsubscript{21} and c\textsubscript{21}. The same process is repeated for LSTM\textsubscript{31}, where h\textsubscript{21} acts as the input vector, to produce h\textsubscript{31} and c\textsubscript{31}. Next, we compute the velocity model at 1.1 Hz using the weights of the fully connected output layer and the vector h\textsubscript{31} as the layer input. It is important to emphasize that while the network’s long-term memory (i.e., c\textsubscript{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. That is, the information stored in c\textsubscript{31} is encoded in the final hidden-state vector h\textsubscript{31}, which we use to compute the velocity model for the current frequency.

Next, we go back to the LSTM\textsubscript{11} cell to demonstrate the horizontal propagation of information within the same layer from lower to higher frequencies. LSTM\textsubscript{12} takes the seismic data for all sources corresponding to the 2.2 Hz input vector x\textsubscript{2}, along with the hidden-state h\textsubscript{11} and the cell-state c\textsubscript{11} vectors of the previous cell in the same layer (which contain information about the velocity being built at 1.1 Hz). It then uses its weights to compute h\textsubscript{12} and c\textsubscript{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 (MSE) between each predicted and true velocity models, which is used to update the weights for each layer through backpropagation.

Due to GPU memory limitations, we twice downsample the velocity models from 500 × 500 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, ∆x = ∆z = 16 m is significantly smaller than the shortest wavelength component in the data. Therefore, this downsampling procedure is expected to have negligible effects on the resolution of the recovered models (i.e., spatial aliasing). 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 using 2D Gaussian convolutional filtering. Figure 3.8 shows the training and validation curves for the network. We use the network weights at 1200 epochs because 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.

3.5 Network Testing

The trained LSTM-RNN is tested using the 5000 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 3.9b and Figure 3.9c show the nine recovered models along with their cross-sections for the testing model in Figure 3.9a. 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 it exhibits all the features of the true model when including the 9.9 Hz data. This behavior is most apparent at the vertical cross-sections. The predicted velocity starts as an average of the two layers; however, as the input frequency increases the predicted model forms two increasingly 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 toward the upper-left corner of the model. While this portion is absent from the sub-5.5 Hz models recovered, its presence is first observed at approximately 5.5 Hz with its shape and size being 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 3.10). The first example shown in
Figure 3.6 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 $h_i$ and $c_i$ outputs at each step $i$ being injected as input at $i + 1$. Increasingly higher frequencies are injected successively along with their corresponding filtered velocity models.
Figure 3.7 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 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.
Figure 3.8 Training (green) and validation (red) learning curves for the LSTM-RNN used in this experiment.

Figure 3.10a 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 3.10b and Figure 3.10c) 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 example (Figure 3.10d-Figure 3.10f), where the network approximates the thin shallow layers with a velocity gradient that accurately captures the low-wavenumber trend. The cross-section in Figure 3.10f shows that the velocity values of the two deeper layers are close, which might explain why the network recovers a single thick layer with an averaged velocity value. The third example (Figure 3.10g-Figure 3.10i) 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 3.10j-Figure 3.10l shows another common form of misfit between true and predicted models. Where 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 LSTM-RNN network to estimate the velocity models for all 5000 testing-data inputs. Because the recovered and true models have different dimensions (i.e., 125 × 125 and 500 × 500 grid points, respectively), we interpolate the
Figure 3.9 (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 3.10 Four examples of velocity models estimated by the trained LRTM-RNN network. (a), (d), (g), and (j) show the true testing velocity models. (b), (e), (h) and (k) present 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 colorbar is used for all velocity models. The normalized root-mean-square (NRMS) misfit between the true and predicted models is 0.023 (b), 0.035 (e), 0.055 (h), and 0.079 (k). These misfit values fall in the 82th, 67th, 42nd, and 22nd percentiles of the NRMS misfit histogram for all 5000 testing data, respectively.
recovered velocities back to their original dimensions of 500 × 500 grid points to perform following quantitative analysis. We then compute the normalized root-mean-square (NRMS) misfit between the true and predicted 9.9 Hz models

\[
NRMS = \frac{1}{2} \sqrt{\frac{\sum (\Phi_{\text{True}} - \Phi_{\text{Pred}})^2}{\sum \Phi_{\text{True}}^2}},
\]

where \(\Phi_{\text{True}}\) and \(\Phi_{\text{Pred}}\) refer to the true and predicted data, respectively. The NRMS measure will be zero for perfectly matched data, equal to unity where \(\Phi_{\text{True}} = -\Phi_{\text{Pred}}\), and be 0.5 when \(\Phi_{\text{Pred}} = 0\). Figure 3.11 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 0.058 (i.e., 5.8%), which helps to quantitatively support and generalize the observations made about Figure 3.10 above.

![Figure 3.11](image)

Figure 3.11 A histogram showing the NRMS misfit between the true and recovered 9.9 Hz velocity models calculated using equation 3.1.

To further analyze trained network performance, we use the recovered models to simulate seismic data using the interpolated 9.9 Hz recovered velocity models following the same modeling procedure discussed above. Figure 3.12a and Figure 3.12c show the associated true and predicted velocity models that have largely consistent long-wavelength components, an observation confirmed by the associated magnitude and phase curves in Figure 3.12d and Figure 3.12e being almost identical at 2.2 Hz. While still being close, the two curves somewhat diverge as the frequency increases to 9.9 Hz. In the second example (Figure 3.13), the recovered model accurately recovers the long-wavelength components of the true model, though it does not fully estimate the velocity of the deeper layer. However, this inaccuracy seems to have little effect on
the seismic data at the highest frequency used in the experiment. This observation reinforces our assertion that the underperformance in the recovery of some models may be caused by poor illumination in the true and recovered data alike.

Figure 3.12 Comparison of the amplitude and phase components of the data simulated using the LSTM-RNN estimated models. (a) True model. (b) Cross-sections through the true and recovered models at the location of the dashed line in (a). (c) Recovered model at 9.9 Hz. (d) Magnitude and (e) unwrapped phase comparisons for data modeled using the models in (a) and (c) at 2.2 Hz, 5.5 Hz and 9.9 Hz. The normalized root-mean-square (NRMS) misfit between the true and modeled data at 9.9 Hz is 0.041 and 0.003 for the amplitude and phase, respectively. These misfit values fall in the 79th and 99th percentiles, respectively, of the NRMS misfit histogram for all 5000 testing data.

In the third example (Figure 3.14), the recovered model well represents the shallower model structure and trends in the correct direction of the velocity inversion at the deeper layer, but it fails to estimate the true velocity. Unlike the previous example, the recovering model inaccuracy has an observable effect on the modeled data in the form of a cycle-skipped unwrapped phase at 9.9 Hz. The fourth example (Figure 3.15) 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 is most likely caused by factors other than the velocity structure. When using these seismograms as input to the LSTM-RNN, the data simulated using the recovered model appear to be a smoother version of the true data.

To better understand how well the ensemble of LSTM-RNN generated models fit the true data, we compute the NRMS misfits (equation 3.1) of the magnitude and unwrapped phase components for data simulated using the true and 9.9 Hz recovered models. Figure 3.16a and Figure 3.16b show the ensemble NRMS of the amplitude and unwrapped phase misfits, respectively. The magnitude and phase error
distributions have means of 7.4% and 4.7%, which are broadly consistent with the observations from Figure 3.11.

3.6 Discussion

The proposed approach is distinguished from other ANN-based methods through its use of discrete frequency slices as the network input (instead of time-domain gathers) and by following a multi-scale strategy commonly used in frequency-domain FWI; however, it suffers from challenges similar to those faced by other NN-based methods, namely scalability and generalization.

Expanding the input data and output velocity model dimensions to real-world 3-D survey sizes remains a significant computational challenge for NN-based velocity model building methods. This is mainly due to model size restrictions caused by the limited global memory available on modern GPU cards. Increasing model dimensionality to the scale of that currently used in industrial 3-D FWI activities would require greatly increasing the network sizes to millions of grid points, and the commensurate increase in trainable parameters would lead to a (currently) prohibitively expensive network training process. Thus, applying NN-based velocity model building at the industrial 3-D FWI scale likely will require the combination of updated GPU hardware along with improved NN training algorithms capable of efficiently embracing
Figure 3.14 Comparison of the amplitude and phase components of the data simulated using the LSTM-RNN estimated models. (a) True model. (b) Cross-sections through the true and recovered models at the location of the dashed line in (a). (c) Recovered model at 9.9 Hz. (d) Magnitude and (e) unwrapped phase comparisons for data modeled using the models in (a) and (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. The normalized root-mean-square (NRMS) misfit between the true and modeled data at 9.9 Hz is 0.035 and 0.161 for the amplitude and phase, respectively. These misfit values fall in the 89th and 1st percentiles, respectively, of the NRMS misfit histogram for all 5000 testing data.
Figure 3.15 Comparison of the amplitude and phase components of the data simulated using the LSTM-RNN estimated models. (a) True model. (b) Cross-sections through the true and recovered models at the location of the dashed line in (a). (c) Recovered model at 9.9 Hz. (d) Magnitude and (e) unwrapped phase comparisons for data modeled using the models in (a) and (c) at 2.2 Hz, 5.5 Hz and 9.9 Hz. The normalized root-mean-square (NRMS) misfit between the true and modeled data at 9.9 Hz is 0.127 and 0.051 for the amplitude and phase, respectively. These misfit values fall in the 11th and 44th percentiles, respectively, of the NRMS misfit histogram for all 5000 testing data.

Figure 3.16 Histograms showing the NRMS misfit between data modeled through the true and 9.9 Hz recovered model calculated using equation 3.1. (a) Magnitude. (b) Unwrapped phase.
large-scale multi-GPU parallelism.

The second major challenge is generalization. The qualitative and quantitative analyses presented herein indicate that the developed network works well when tested on models not used in the training process. However, we note that the testing models were generated using the same engine that generated the training data and thus the testing data used cannot be considered as “out of sample”. Further work is needed to make the network more general. This includes enabling it to recover velocity structures not included in the training data (i.e., “out of sample” models), and to handle frequency slices different than those used in the training process.

3.7 Conclusions

We develop a novel ANN technique that is inspired by the FWI multi-scale 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 where the degree of model smoothing depends on the frequency being injected at that RNN step. The presented examples along with quantitative analysis demonstrates that the LSTM-RNN network is capable of recovering relatively complex velocity models with high accuracy. These results are corroborated by independent forward modeling tests that show data simulated with the true and 9.9 Hz recovered models match well. This observation holds even for models that appear somewhat dissimilar when viewed in the velocity model domain, which means that much of the velocity differences likely fall within the null space of the experiment.
4.1 Abstract

Artificial neural networks are emerging as an alternative approach for seismic velocity model building that does not inherently suffer from the challenges associated with deterministic methods. They do, though, have their own set of drawbacks, chief among which arguably is the difficulty of network generalization. While generalizing to testing models with structures similar to those found in the training data has become a manageable task, extending generalization to more realistic scenarios where testing models may have drastically different velocity structure and/or distribution than those in the training data set remains an important and ongoing research. To address this issue, we present a multi-scale approach inspired by full-waveform inversion that uses recurrent neural networks to invert frequency-domain seismic data using a frequency-stepping scheme. We combine this approach with a hybrid training approach that combines background velocity gradient models with purely geometrical and geologically realistic model structures. We demonstrate the potential for improved generalization by comparing the model estimates results from two trained networks: one using a hybrid set of models, the other with only geological models. We test the two networks using BP2004 benchmark model subsets with complex salt structures fully absent from the models used in the training process. Qualitative and quantitative analysis shows that models recovered using hybrid training data are significantly more accurate than those recovered using geological training data alone, with arbitrarily-shaped salt bodies being accurately delineated by the trained hybrid network.

4.2 Introduction

Seismic velocity model building is one of the most challenging tasks in seismic exploration. Traditionally, the problem has been approached using deterministic methods that either use a subset of the recorded seismic waveform (e.g., arrival picks in traveltime tomography) or the full waveform itself (e.g., full waveform inversion [FWI]). The aim of FWI algorithms is to build a high-resolution subsurface velocity model that minimizes the misfit between observed and modeled seismic data [7, 8]. The greatest FWI challenge, though, is that the governing wave equation is non-linear with respect to model parameters, which leads to an inversion problem with a highly non-linear objective function with numerous local minima. Such issues are exacerbated by the oscillatory nature of seismic data, which demands using starting earth models that when used in seismic forward modeling can simulate waveforms to within half a
period of recorded data. This requirement helps to avoid cycle skipping that can cause 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.

A number of approaches have been proposed in the past decades to overcome the non-linearity challenges associated with the seismic velocity inversion problem. For example, many FWI approaches follow a multi-scale strategy [11, 36, 57, 58] that first inverts lower-frequency data to promote the recovery of smoother velocity models (i.e., longer-wavelength structure), which ideally moves the optimization search direction toward the global minimum. The updated smooth velocity models are then input to FWI engine to invert higher-frequency data and increase model accuracy and resolution.

The recent resurgence of neural networks, though, 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 model building methods. ANNs offer the potential of improved efficiency because the main cost overhead comes from ANN training, after which model predictions can be made at the cost of matrix-vector multiplication. There have been numerous successful applications of ANNs for solving the seismic velocity model building problem, including inverting common mid-point semblance cubes [39] and using different neural network architectures to invert data [17, 22–24, 54, 59]. A common feature in these approaches is the use of time-domain data as network input, which requires gathers to be input simultaneously into the network.

In this work, we propose a novel frequency-domain ANN approach that uses a recurrent neural network (RNN) to feed frequency slices incrementally rather than globally from the lowest available to highest usable. While this approach reduces the amount of data needed to estimate the underlying velocity model, it still suffers from the common challenge of generalization which, in this context, means the following: if a network is trained on a training data set and then is successfully tested on data drawn from the same distribution not used in the training data set, the network is said to have generalized well. This definition of generalization, though, is of limited use in the seismic inversion context because in most cases one does not have the local geological information required to build training models sufficiently representative of those being estimated.

To help minimize generalization issues, we present a novel training approach that combines a FWI-inspired frequency-stepping approach with a hybrid velocity model training scheme based on three velocity model classes: background, geometric, and geologic. We show that compiling and training ANNs with a “hybrid” data set derived from the hybrid modeling approach leads to improved model estimation and a corresponding improvement in network generalizability. We begin by presenting our procedure for
constructing realizations of each training model class. We then describe the neural network architecture used in this work and explain our rationale behind this choice. Next, we outline the training process and show the results of testing the proposed network using subsets from the BP2004 benchmark model and discuss general trends in the recovered models. We subsequently discuss the computational challenges in applying the proposed method to real-world survey sizes as well as the challenges introduced when aiming to invert field data.

4.3 Generating Training Data

To demonstrate the benefits of combining different velocity model classes in the network training, we first need to specify the procedures used to generate models of the three distinct classes: background, geometric, and geologic.

4.3.1 Background Models

The first set of models are intended to represent background geologic velocity trends, such as the compaction-related tendency - but not universality - of positive velocity gradient with increasing depth. Figure 4.1a presents examples of background models, which include those with a constant velocity as well as velocity gradients with randomly oriented dips. We randomly specify both whether the velocity increases or decreases with depth as well as the background velocity value and starting and ending velocity gradient values.

4.3.2 Geometric Models

The second training model class is purely geometric, examples of which are illustrated in Figure 4.1b. Alzahrani and Shragge [60] demonstrate that purely geometrical training models are not biased toward specific geologic structures, which makes them useful in situations where geological structures in the testing models cannot be predicted. Instead of using squares as in Alzahrani and Shragge [60], here we construct our purely geometrical models using a more elaborate process that involves a sequence of random numbers to promote continuity of the generated random structures. We start by randomly specifying the number of squares in each direction. To understand how velocities are assigned to each square, imagine an ant is dropped on one square that takes a random walk until reaching the domain boundary. A randomly selected velocity is then assigned to each square along the random path on which it stepped. This process is repeated until all squares have been assigned a velocity, with the ant each time starting from a square not yet assigned a velocity value. This process greatly increases the likelihood that the resulting velocity model consists of continuous structures and not just randomly scattered velocity values. Another sequence of random processes determines the degree and orientation of the applied model
Figure 4.1 Representative 1.25 km × 4.0 km examples drawn from the three training model classes. (a) Geological models with faulted layered structures. (b) Purely geometric models at different scales with variably oriented structures. (c) Background models with either a constant value or a gradient in a uniformly distributed random orientation.
smoothing operators to ensure the structures at different scales point in varying directions (see Figure 4.1b).

4.3.3 Geologic Models

For the final class, we construct geologically inspired models from top to bottom in a number of steps illustrated in Figures 4.2 and 4.3. First, the number of model layers is randomly specified within a certain range. Then, the first interface is delineated using a number of equidistant control points, the number of which as well as their depth locations are randomly specified within a range of accepted values. The blue dots in Figure 4.2a show the control points for the top layer, which are permitted to follow the topology of any geological surface. The interface is then constructed using linear interpolation between these control points (Figure 4.2b). The control points of the second interface are then specified. The horizontal locations of these points are the same as those of the first interface, while their depth is constrained to keep the layer thickness within a certain range (Figure 4.2c). Linear interpolation is then used to construct the second interface (Figure 4.2d). These two constraints prevent the second layer from developing completely different structure from the overlying layer. These steps are repeated for all subsequent layers.

After constructing all layer interfaces, we use a 1-D Gaussian operator to smooth the piece-wise linear interfaces (Figure 4.2e). A two-step process was deemed necessary because linear interpolation produced sufficient and stable interpolation results, especially when using a large number of control points. Subsequently, because a piece-wise linear interface is not geological, we smooth it using 1-D Gaussian operator. Velocity values are then randomly assigned to each layer. Following this, a randomly generated number specifies whether the current model should include faults as well as the vector offset such a fault should exhibit. The fault line is drawn by randomly specifying and interpolating between two points located at the top and bottom boundaries (see Figure 4.3a), with the layers shifted by a randomly specified offset (see Figure 4.3b). The number of faults in the model (inclusively between one and three) is randomly specified as well. If a model has multiple faults, the same fault line is duplicated at different randomly specified model locations (see Figure 4.3c) and the same offset is applied at all fault locations (Figure 4.3d).

4.3.4 Training Data Sets

We generated 6000 geologic models, as well as 2000 models for each of the geometric and background model classes. Furthermore, 2000 models were extracted from the set of geological models and combined with the other 2000 models generated for each of the other two classes to form a hybrid data set of 6000
Figure 4.2 Illustration of the five steps used to generate a three-layer model. (a) The control points of the first interface are randomly specified followed by (b) a linear piece-wise interpolation between these points. (c) The control points of the second interface are specified followed by (d) a linear interpolation. (e) 1-D Gaussian smoothing operator is applied to the constructed interfaces. (f) Velocity values are assigned to the resulting layers.
Figure 4.3 Illustration of the model faulting process. (a) A fault line is specified by interpolating a straight line between two randomly located points at the top and bottom of model boundary. (b) Layers are shifted at the fault line by a randomly specified amount. (c) A model with three parallel fault line. (d) The layers are shifted at the fault line locations by the same vector offset.
models.

We simulated seismic data for each model in the three classes by solving the 2-D constant-density acoustic wave equation in the frequency domain (Helmholtz equation) using a 15 Hz Ricker wavelet. We solved the Helmholtz equation using an implicit finite-difference (FD) method with the solution mesh discretized at 10 m in each direction. Data were generated at 38 source locations with a 200 m shot spacing for six frequencies ranging from 1.0 Hz to 6.0 Hz in 1.0 Hz increments. Sources and receivers were respectively located two and five grid points below the free surface. The monochromatic wavefield response was recorded as receiver data every 10 m across the breadth of the model.

4.4 Neural Network Architecture

Convolutional neural networks (CNNs) have been used to successfully invert time-domain seismic data. However, when using CNNs to invert frequency-domain data, a natural procedure would be to feed each frequency slice to a different channel of the input layer. A problem with this approach is a lack of communication between different frequency slices, which precludes using it to exploit a multi-scale FWI strategy where one first constructs a smoother velocity model from lower-frequency data and then adds in finer details from higher-frequency data.

RNNs are one class of ANNs that allows information flow from prior input stages [52]. These architectures incorporate a hidden-state vector that acts as network memory and carries information “learned” from one input to the next, which makes this ANN class highly prospective for exploiting a multi-scale FWI strategy. RNNs can be classified into two categories based on their type of constituent layers: conventional and convolutional (see Figure 4.4). Because RNNs handle two vectors/feature maps at each step, an RNN layer actually consists of two layers: one handling the input data, the other the hidden-state vector. As illustrated in Figure 4.4, the only difference between conventional and convolutional RNNs is that the former uses fully connected layers while the latter uses two convolutional RNN layers.

Convolutional RNNs combine the advantages of CNNs (including a significant reduction in required memory) that have made them successful in a wide range of applications, while simultaneously benefiting from including hidden-state feature maps to propagate information from the initial to the final input. RNNs may also be classified based on how they propagate information between steps. Among these different classifications, long short-term memory (LSTM) networks are the most suitable for the present application due to their stability for long sequences. This is because RNNs are susceptible to the vanishing or exploding gradient problem, which makes them incapable of carrying forward information learned for long sequences [56]. The LSTM network circumvents these issues by incorporating four layers (each with two constituent layers for eight in total) that uses two memory vectors to propagate information between
Figure 4.4 Comparison between conventional and convolutional RNNs. (a) The two fully connected layers comprising the conventional RNN. Information flows between step by adding the output of the hidden-state layer to that of the layer handling the input data. (b) The same process as in (a) but with convolutional operators connecting the input and output instead of connective weights. Items with the same role are indicated with the same color.
steps (Figure 4.5b). This network structure is in contrast with the two layers and one memory vector found in conventional RNNs. The fully connected and convolutional RNNs respectively illustrated in Figure 4.4a and Figure 4.4b propagate information using the logic shown in Figure 4.5a. They simply add the output of the two layers processing the input data and the hidden state-vector/feature maps after each step. The additive step replaces the hidden state vector/feature maps after each step, which causes information to usually be completely lost after three or four steps. This is the reason that among the four layers (gates) constituting an LSTM cell: the “forget layer” performs the crucial task of acting as a gate for information flow between steps. Because the network has limited memory, it must discard information deemed unnecessary, which allows relevant information to reach the end of the sequence.

We use a convolutional LSTM neural network architecture that we henceforth refer to as ConvLSTM. Specifically, we use a sequence-to-sequence ConvLSTM, in which each input is constrained by an output with the input-output connection guiding the training process (Figure 4.6). Following the logic of multi-scale FWI approaches, we first inject the lowest usable data frequency into the network. Ideally, the output velocity model has wavenumber components not much greater than those found in the forward modeled wavefield from the injected source frequency. However, because constructing such model is not straightforward for heterogeneous media, we approximate this process by applying 2-D Gaussian smoothing at each step with the convolutional operator width dependent on the input frequency. We use a wider operator for lower frequencies to relatively upweight the longer-wavelength components sensitive to model structure at the current frequency. After injecting the smooth velocity model as the output for the first frequency, we proceed by following the similar process for higher frequencies, but smoothing with narrower
2-D convolutional Gaussian filters. For a more detailed description of the network dynamics, we refer readers to Alzahrani and Shragge [61].

Figure 4.6 A sketch showing the network architecture used in this work. The network consists of three ConvLSTM layers. Information flows from one frequency to the next via two memory vectors. Each input frequency is constrained by the corresponding 2-D Gaussian-smoothed velocity model.

The network used in this work consists of three ConvLSTM layers (see Figure 4.7). The input monochromatic frequency slice contains data from all sources and receivers and is arranged in a 2-D array with 76 elements along the vertical axis representing the real and imaginary components of the 38 sources used in the experiment and 400 elements along the horizontal axis corresponding to the number of receivers. In a convolutional LSTM network each layer has the same hyper-parameters as a typical convolutional layer; namely, the size and number of 2-D convolutional operators. Figure 4.7 depicts our parameter choices for each layer. We observe that applying convolutions with a stride larger than unity produces more stable results than using pooling layers. To downscale the input data along the receiver axis, we use stride values of six, four, and two for the layers ConvLSTM1, ConvLSTM2, and ConvLSTM3, respectively. These values strike a balance between model resolution and computational efficiency. Each ConvLSTM layer is followed by a batch normalization layer to reduce overfitting and improve network generalization. The 2-D feature maps corresponding to ConvLSTM3 are flattened into a fully connected layer that is in turn connected to the output velocity layer.

4.5 Training

Training the ANN involves inputting data to the network, performing a forward run to compute the outputs, and using the difference between true and predicted output to update the network weights.
Figure 4.7 Convolutional parameters for each network layer. The input data frequency slice consists of 38 sources and 400 receivers with the top and bottom 38 rows consisting of real and imaginary data components. The first two numbers below each ConvLSTM layer represent the 2-D convolutional filter size in the vertical and horizontal directions, the last number is the number of filters, and their product is the number of output feature maps for each layer. The output 2-D feature maps for ConvLSTM3 are flattened into a vector for connection to the fully connected output velocity model layer.

through backpropagation. Because we use a sequence-to-sequence network (see Figure 4.6), performing a forward run produces a velocity model for each input frequency. To ensure that the network only fits structures at the appropriate scale for the input frequency, we use a structural similarity index measure (SSIM) loss function with a variable (i.e., frequency-dependent) analysis window. That is, we use larger SSIM windows for lower frequencies and decrease the window size with increasing frequency (Figure 4.8) as determined through empirical testing. We use unsmoothed output velocity models for the loss-function analysis to enable the different SSIM window sizes to capture different wavenumber trends; however, SSIM implementation internally applies Gaussian smoothing using an operator length dependent on the SSIM window size. Therefore, while the output velocity models are not explicitly smoothed, the network still follows the logic illustrated in Figure 4.6 with the smoothing being performed implicitly in the loss function instead of being explicitly performed before training.

To measure the value added to the training process by hybrid data, we train two networks with the same architectures and hyperparameters but different training classes: (1) Network $G$ with purely geologic models, and (2) Network $H$ combining hybrid background, geometric and geologic data. The validation data serve as a measure of the neural network’s ability to generalize learning to unseen data. During the training process, we test each network on a set of velocity models not included in its training. The validation data sets for both networks consist of all three training data types. By comparing network performance on training and validation data, we can determine whether the network is over- or underfitting. A decreasing training loss with increasing validation loss suggests that the network is
Figure 4.8 Illustration of the SSIM loss function used for network training. The SSIM performs a window-to-window (instead of point-to-point) comparison between true and predicted velocity models. The squares in each image represent the decreasing window size used to measure the fit between true and predicted velocity models with increasing frequencies.

overfitting and has learned patterns specific to the training data rather than generalizing its learning. Conversely, a constant validation loss may indicate that the network is not effectively learning and the process should be stopped to minimize computational cost.

Figure 4.9a and Figure 4.9b show the learning curves for Networks $G$ and $H$, respectively. Hybrid validation data were used for both networks. The SSIM training loss function outputs a normalized value ranging between 0 (least similar) and 1 (identical). Because SSIM measures similarity and not misfit, we formulate the loss function as $1 - \text{SSIM}$. Minimizing such a loss function effectively maximizes the similarity between true and predicted models, which is the reason why the curves in Figure 4.9 start close to unity at the first training epoch and decrease toward zero as learning progresses. Based on the curves presented in Figure 4.9, we elect to stop training after 50 epochs. While the validation curves continue to slowly decrease, our qualitative analysis indicates that at later epochs the network fits some model components better, but at the cost of introducing short-wavelength noise at other locations. Additionally, because we aim to improve network generalization beyond the model characteristics included in the validation data, we prefer to under- rather than overfit the trained networks.

4.6 Testing

To investigate the degree of generalization of each network, we test their respective performance using the BP2004 model [37] shown in Figure 4.10. Using the full BP2004 model of dimension $1911 \times 5359$ grid
Figure 4.9 Training and validation curves for Networks (a) $\mathcal{G}$ and (b) $\mathcal{H}$. 
points for testing purposes currently is not possible due to GPU memory limitations. Therefore, we extract model subsets of size 500 $\times$ 1600 (i.e., rectangles in Figure 4.10) and downscale them by subsampling by four in each direction to the desired network output size of 125 $\times$ 400. Through empirical testing, these model dimensions proved to be of sufficient size to capture the main structural elements within individual testing models. Figure 4.10 also shows the locations of neighboring horizontal (white rectangles) and vertical (purple rectangles) model subsets. We use a 50% point overlap between adjacent subsets in both directions, which allows 6 $\times$ 5 subsets to be extracted from the full BP2004 model. We note that none of these 30 models were used in the training process and some exhibit significantly different structures and velocity distributions from those found in the training models. Therefore, the predictive performance on these subsets should represent a good test of how well each network has generalized.

![Figure 4.10 The BP2004 model with white and purple rectangles showing the respective locations of horizontally and vertically adjacent model subsets.](image)

Figure 4.11 shows the recovered velocity models using the network trained on geological models only. Instead of separately investigating each of the 30 subsets, we “patch” each recovered subset in its true model location and compute the mean velocity value in areas of overlap. We emphasize that forming the full BP2004 model by merely concatenating different subsets does not mean we are claiming that either network can globally recover the full BP2004 model (which would be prohibitively expensive at present). Specifically, such a test would require placing sources and receivers only at or near the free surface as opposed to throughout the model as is done in the present experiment. Rather, our goal is to provide an initial visual evaluation of network performance and to test the consistency of that performance using the range of geological characteristics exhibited by the BP2004 model.

Figure 4.11a shows the recovered subsets using network $G$ patched back into their original locations. Given the clear footprint of the model subsetting procedure in this result, we apply 2-D Gaussian smoothing (see Figure 4.11b) to recover more continuous geologic trends that better resemble a similarly smoothed BP2004 model. Evidently, the models recovered by Network $G$ suboptimally reconstruct the salt body geometries and velocity values, as well as the shape of the sedimentary layer overlying the salt.
especially in the immediately overlying units. Overall, Network $G$ exhibits a limited ability to recover structures that are neither flat nor gently dipping; however, we note that some dipping structure can be introduced through the 2-D Gaussian smoothing post-processing operation. Figure 4.12a and Figure 4.12b respectively show the recovered subsets using Network $H$ patched back into their original locations without and with 2-D Gaussian convolutional smoothing applied. Compared to the Network $G$ results, the models recovered using Network $H$ more closely resemble the true velocity profiles. Most model components have been recovered with improved accuracy including the two salt bodies and the enveloping sediment. Even before smoothing the separate subsets in Figure 4.12a, the image exhibits improved overall continuity than that recovered by Network $G$. Moreover, the geological structures are more accurately recovered within each subset as opposed to being introduced as a smoothing artifact. For example, the steep salt dome flanks are more accurately recovered even before smoothing.

![Image](image.png)

Figure 4.11 The 30 BP 2004 model subsets recovered using network $G$ patched into their true locations (a) before and (b) after smoothing.

The patching process (i.e., concatenation, averaging, and smoothing) to form the full BP2004 model estimates in Figure 4.11 and Figure 4.12 causes loss of details. Thus, we zoom into areas of interest in individual model subsets for more detailed information about the networks’ relative (de)merits. Figure 4.13 shows a subset capturing two different salt dome areas. The white rectangles in Figure 4.13a indicate the locations and extents of the two subsets. Figure 4.13b and Figure 4.13c shows the true velocity model extracted at these locations. Figure 4.13d and Figure 4.13e (Figure 4.13f and Figure 4.13g) present the subset models recovered using Network $G$ (Network $H$).
Figure 4.12 The 30 BP 2004 model subsets recovered using network $H$ patched into their true locations (a) before and (b) after smoothing.

Figure 4.13 Comparison between models recovered by Networks $G$ and $H$ for subsets indicated by the white rectangles in (a). The true velocity models extracted at the location indicated by the (b) solid and (c) dotted lines. (d) and (e) The two subsets recovered using Network $G$. (f) and (g) The two subsets recovered using Network $H$. 
The results presented in Figure 4.13 again suggest significant differences between the models recovered by Networks $\mathcal{G}$ and $\mathcal{H}$. In both subsets, the velocity models recovered by the latter provide more accurate representations of the true model than the those recovered by the former. For the subset shown in Figure 4.13d, Network $\mathcal{G}$ model estimates do not resemble the true model because no recovered structure appears at the model center; instead, the network has estimated a velocity model comprised of three layers with a smooth increase in the lower layer’s relief around the model center. Conversely, Network $\mathcal{H}$ model estimates (Figure 4.13f) depict a more accurate reconstruction of the salt body and surrounding area. The salt dome geometry is accurately approximated with the right dome flank being raised higher than on the left and the velocity structures to either side are accurately recovered. In addition, the transition between layers of strong velocity contrast is more gradual, likely as result of smoothing training models prior to injection as network input. This allows the salt body to accurately reconstruct velocities at a distance from the boundary, as shown above. The performance of each network in the second subset is consistent with that of the previous example. Network $\mathcal{G}$ (Figure 4.13e) fails to accurately reconstruct the salt body, while Network $\mathcal{H}$ (Figure 4.13g) well recovers both the salt dome and the surrounding sediment.

The second example examines network performance in recovering the more challenging leftmost BP2004 model salt body. Before discussing the results of estimating the entire salt body, we first examine smaller subsets formed by subsampling by a factor two instead of four as above (i.e., model subsets of dimension $250 \times 800$ instead of $500 \times 1600$). This allows for different complex salt body structures to be extracted. Figure 4.14 shows two subsets with different salt structures with the same plot layout as used in Figure 4.13. In both subsets, the Network $\mathcal{H}$ performance in estimating true salt structure is superior to that of Network $\mathcal{G}$. Note that the highly smooth Network $\mathcal{G}$ model recovery in Figure 4.13 and Figure 4.14. We hypothesize that Network $\mathcal{G}$ may perform well on some non-layered models because the smoothing operators applied during the training process effectively can transform a training geological model to a smooth geometric training model. Therefore, Network $\mathcal{G}$ implicitly incorporates semi-geometrical training models at lower frequencies.

Figure 4.15 presents the final subset example for two of the arguably most complex BP2004 model components using the same layout as the previous two figures. Again, Network $\mathcal{H}$ (Figure 4.15f and Figure 4.15g) significantly outperforms Network $\mathcal{G}$ (Figure 4.15d and Figure 4.15e) in both subsets. However, as the salt body geometry becomes more complex, a number of patches with velocities slower than the true values appear at various salt body locations on the Network $\mathcal{H}$ recovered model. We speculate that this may be due to geometric training data consisting of intersecting geometric shapes with widely varying velocity values. Given the usually large velocity variations of these intersecting shapes, Network $\mathcal{H}$ does a good job in keeping recovered velocity estimates to within a close range.
Figure 4.14 Comparison between models recovered by Networks $\mathcal{G}$ and $\mathcal{H}$ for subsets indicated by the white rectangles in (a). The true velocity models extracted at the location indicated by the (b) solid and (c) dotted lines. (d) and (e) Subsets recovered using Network $\mathcal{G}$. (f) and (g) Subsets recovered using Network $\mathcal{H}$. 
(< 0.5 km/s) of the true values. We selected these two subsets to show the consistency of Network \( \mathcal{H} \) performance when the extracted model subsets are only slightly shifted from each other. We observe a high degree of similarity between the recovered models, which suggests a reasonable stable network model estimation process.

Figure 4.15 Comparison between models recovered by Networks \( \mathcal{G} \) and \( \mathcal{H} \) for subsets indicated by the white rectangles in (a). The true velocity models extracted at the location indicated by the (b) solid and (c) dotted lines. (d) and (e) Subsets recovered using Network \( \mathcal{G} \). (f) and (g) Subsets recovered using Network \( \mathcal{H} \).

To quantitatively evaluate network performance, we first generate a suite of 2200 model subsets extracted at increments of 50 grid points in both directions. Each subset is initially of dimension 500 × 1600 and then subsampled to fit the 125 × 400 network size. Due to the significant geological variability in the BP2004 model, neighboring subsets may contain variable geological structure that present different challenges to the trained networks and potentially exhibit a variety of seismic signatures than when compared to the previous examples. The next step requires defining an evaluation metric to provide a quantitative measure of the degree to which the model subsets are recovered by each network. Because computing pixel-by-pixel differences between true and predicted models using differential metrics (e.g., root-mean square error) do not capture the structural accuracy of recovered velocity models, herein we use a SSIM statistical metric to compare true and predicted models over a window of user-defined size. Given the smooth nature of recovered models, we chose sliding windows of dimension 51×51 (500 m by 500 m) to
conduct SSIM analyses, which based on empirical testing provided a representative measure of recovery accuracy.

Figure 4.16 shows two histograms representing the distribution of SSIM values for each network. The mean and median SSIM values corresponding to Network $H$ are 0.1 higher than those of Network $G$, while its standard deviation is 0.05 lower. Overall, Figure 4.16a shows that Network $G$ model accuracy is almost equally distributed on both sides of the mean, which is located at the distribution center. However, Figure 4.16b shows that the large majority of Network $H$ models fall between 0.5 – 0.8. We speculate that the accuracy drop in the recovery of Network $H$ models is due to the presence of low-velocity regions within recovered salt bodies.

4.7 Discussion

The proposed method has two main limitations stemming from a limited capacity to: (1) handle higher frequencies, and (2) recover large models with real-world survey sizes, both of which are due to limited computational memory capacity. To address these computational limitations, we performed aggressive striding in the convolutional step of each layer to fit the network in the NVidia V100 GPU global memory. These actions resulted in the loss of high-frequency information as the network analysis progressed to the output layer. This also precluded a globally estimation of the full BP2004 model and led to our procedure...
of extracting smaller model subsets, inverting these with the trained networks, and then patching them to the appropriate model location. Consequently, these tests effectively placed sources and receivers at different depths and lateral locations, a requirement not physically realizable in field settings. Introducing larger GPU global memory sizes or computing paradigms to easily distribute training/testing processes across multiple GPU resources would permit testing with only near-surface sources and receivers.

Although the ANN discussed herein is inspired by multi-scale FWI approaches, it was not designed to supplant this analysis. On the contrary, we assert that the long-wavelength nature of the recovered RNN models enables them to serve as starting FWI models, which could reduce the number of required iterations and hasten convergence of multi-scale FWI applications. In addition, because the recovered models approximately reconstruct subsurface geometries and associated velocity values, they can be integrated with other information (e.g., well logs, tomography outputs) used to construct FWI starting models. They also could be used in reconnaissance 2-D seismic surveys to develop initial subsurface models before proceeding with 3-D seismic data acquisition or related analyses.

Finally, while representing a promising method of velocity estimation, the current proposed network would need significant further testing to move beyond synthetic data and be robust for field data applications. For example, one should investigate how the network behaves in the presence of different types of random and coherent noise that are ubiquitous in field data. In addition, network sensitivity to the different seismic processing workflow steps used to condition seismic data for imaging and inversion needs to be assessed. Overall, these topics are important research areas for ensuring the applicability of this technology in field data settings.

4.8 Conclusions

We develop an ANN scheme that mimics the FWI multi-scale approach that uses hybrid background, geometric, and geologic training data to improve network generalization. Instead of adopting the prevalent practice of generating highly realistic geological models, we augment our straightforward layered geological models with random geometric shapes and background gradients that capture a range of model statistics commonly exhibited in earth models. I test the value added when using hybrid instead of geological-only training models by training two networks with the same architecture and hyperparameters using different training data classes: one trained using geological-only models, the other with a hybrid of background, geometric and geological models. Our testing results on the complex BP2004 velocity model not used in the training process support the assertion that hybrid training data can improve network generalization by introducing a large number of random structures at different scales. I demonstrate through a SSIM metric analysis that the developed RNN approach complements the geological training data and extends the range.
of structures recoverable by the trained ANN.
CHAPTER 5
CONCLUSIONS

Due to the highly non-linear relationship between seismic data and subsurface velocity, deterministic methods face limitations in their ability to solve the velocity model building problem. In particular, the more successful approaches rely on iterative inversion-based methods that required an initial velocity model with accurate long-wavelength components (e.g., multi-scale full waveform inversion or FWI). Conversely, neural networks have been demonstrated to be effective in addressing the seismic velocity model building problem, without suffering from the same challenges that hinder deterministic methods. However, neural networks are not without their own challenges. Foremost among these is an ability to generalize to out-of-distribution testing models, such as is the case in areas where the geology is unknown. Addressing such a significant challenge is essential before NNs could be used in realistic velocity model building situations from field seismic data.

This thesis develops a novel ANN scheme that is inspired by the FWI multi-scale approach that uses a sequence-to-sequence convolutional RNN. The input sequences consist of frequency-domain seismic data ordered by frequency from lowest available to highest selected, while the corresponding output sequences are frequency-dependent smoothed velocity models. I significantly enhance its ability to generalize by employing multiple classes of training models that increase the variability of structures as well as wavenumbers present in the data used in the training process. I test the developed network using models containing arbitrarily shaped salt structures that are significantly different from those found in the training data. Qualitative and quantitative analyses demonstrate that the network is capable of recovering complex salt structures with high accuracy.

In 2, I demonstrate the potential in harnessing the power of ANNs to solve the seismic velocity inversion problem. The developed approach does not require any background velocity models to be provided as input to the CNNs, and all models were predicted based on minimal information after network training. This property of ANNs could be significant for mitigating the often severe non-linearity problem of conventional seismic velocity inversion methods. Because neural networks are data-driven tools, though, their performance is mainly controlled by the data used in the training process. While we do not need to provide an ANN with all possible velocity models for it to recover meaningful seismic velocity inversion results, we must still provide it with a representative and sufficient distribution of models during the training process. My analysis provides insight on what features or structures need to be included in the velocity model training data and how the inclusion or exclusion of such structures can affect overall network performance.
I generated four sets of training velocity models: two “semi-geological” and two purely geometric, both with and without velocity gradients. Each test model revealed the advantages and limitations of a particular choice of training data. The models recovered using semi-geological training data were more continuous and exhibited less unwarranted short-wavelength velocity variations. Scenarios where the testing models included velocity gradients or a sequences of thin layers of uniform or slowly varying velocity required semi-geological models with velocity gradients to recover a sufficiently accurate estimate. Conversely, networks trained using purely geometric data successfully recovered continuous geologic structures, especially in the near surface, even though such structures were completely absent from their training data. This intriguing result demonstrates the remarkable capability of ANNs in solving the velocity model building problem using a number of discrete frequency slices. In addition, these results suggest that purely geometric models could be used as a class of training data that is not explicitly biased toward specific geologic structures.

While the CNN used in 2 could recover velocity models with moderate accuracy, its main weakness is a lack of communication between the different frequency slices in the input layer. Specifically, low- and high-frequency slices are fed to the network as different channels in the input layer and are processed separately. This precludes exploiting the multi-scale FWI approach. To relax this restriction, 3 develops a novel ANN technique inspired by the multi-scale FWI approach that uses sequence-to-sequence RNNs. This approach successively injects higher-frequency data slices at each step. To constrain the information carried through the network by the RNN’s hidden state vector, I also inject a smoothed velocity model as the output for each input frequency where the degree of model smoothing depends on the frequency being injected at that RNN step. The presented examples along with quantitative analysis demonstrate that the RNN network is capable of recovering relatively complex velocity models with high accuracy. These results are corroborated by independent forward modeling tests that show data simulated with the true and up to 9.9 Hz estimated models match well. This observation holds even for models that appear somewhat dissimilar when viewed in the velocity domain, which means that much of the velocity differences likely fall within the null space of the experiment.

The developed network in 3 works well when tested on models generated using the same engine that generated the training data. However, extending its generalization abilities to models containing velocity structures not present in the training data remains a challenge. In 4, I develop an ANN scheme that mimics the FWI multi-scale approach that uses a hybrid class consisting of background, geometric, and geologic training data to improve network generalization. Instead of adopting the prevalent practice of aiming to generate highly realistic geological models, I augment my straightforward layered geological models with random geometric shapes of varying spatial wavelengths and background gradients that
capture a range of model statistics commonly exhibited in earth models. I test the value added when using hybrid instead of geological-only training models by training two networks with the same architecture and hyperparameters using different training data classes: one trained using geological-only models, the other with a hybrid class of background, geometric and geological models. My testing results on the complex BP2004 velocity model not used in the training process support the assertion that hybrid training data can improve network generalization by introducing a large number of random structures at different scales. Through a SSIM metric analysis, I demonstrate that the developed RNN approach complements the geological training data and extends the range of structures recoverable by the trained ANN.

5.1 Future Work

This work represents a first step in exploring neural network’s ability to generalize to out-of-distribution testing models when solving the velocity model building problem. Further work is needed before the developed network can be used for field data applications. This includes testing its robustness in the presence of different types of noise in the input data, investigating how the different seismic processing steps affect the network’s ability to generalize. In addition, more work is needed to explore how to best synthesise the seismic training data, such as assessing whether acoustic modeling is sufficient to invert field data, and how sensitive the network to changes in the seismic wavelet.

Furthermore, the design of the geometric training data needs to be improved to eliminate the slow-velocity regions in recovered complex salt bodies. An in-depth investigation of why geometric training data improve generalization could help in improving their design. This could lead to either adding more constraints on the process generating the geometric training models, or allowing for more freedom to increase the variability in their structures. For example, this could shed light on whether restricting the overlap between the geometric shapes would improve the network’s accuracy.

Moreover, additional work is required to address the computational challenges of expanding the input data and output velocity model dimensions to real-world 3-D survey sizes. Currently, their sizes are limited by the global memory available on modern GPU cards. The fully-connected layer responsible for generating the output velocity model is the main limiting factor in the current implementation, as it consumes a significant amount of memory. The network design could be made more efficient by experimenting with different decoding architectures that could replace the fully connected layer.
REFERENCES


APPENDIX
PERMISSION FROM CHAPTERS

A.1 Permission from Journals

This appendix includes the copyright permissions from Geophysics (Figure A.1) and IEEE (Figure A.2) for use of the published papers incorporated as chapters in this thesis. Copyright permissions from Geophysics are detailed in the URL https://library.seg.org/page/policies/open-access. Copyright permissions from Geophysics are detailed in the URL https://journals.ieeeauthorcenter.ieee.org/choose-a-publishing-agreement/avoid-infringement-upon-ieee-copyright/.

![Traditional publication (including green open access)](image)

- No author publication charge (APC) is levied, although payment of mandatory page charges are assessed and payment of voluntary charges is requested. Relief from mandatory charges may be requested under SEG's hardship relief policy.
- Copyright is transferred to SEG.
- Authors/employers retain proprietary rights such as the right to patentable subject matter and the right to make oral presentation of the work with full citation and proper copyright acknowledgment.
- Authors/employers enjoy the right to prepare and hold copyright in derivative publications based on the paper provided that the derivative work is published subsequent to the official date of the original paper's publication by SEG.
- Authors/employers may post a final accepted version of the manuscript or the final SEG-formatted version (book chapters excluded) on authors' personal websites, employers' websites, or in institutional repositories operated and controlled exclusively by authors' employers provided that:
  1. the SEG-prepared version is presented without modification;
  2. copyright notice and a full citation appear with the paper;
  3. a link to the SEG version of record in the SEG Library using the digital object identifier (DOI) and a permalink is provided;
  4. the posting is noncommercial in nature, and the paper is made available to users without charge; and
  5. that notice be provided that use is subject to SEG terms of use and conditions.
- Authors/employers may not post their articles in an institutional repository or other site in which the content is required to carry for example, as carrying a license contrary to SEG copyright and terms of use and terms of this policy.

Figure A.1 A screenshot granting permission from Geophysics.
Can I Reuse My Published Article in My Thesis?

You may reuse your published article in your thesis or dissertation without requesting permission, provided that you fulfill the following requirements depending on which aspects of the article you wish to reuse.

- **Text excerpts:** Provide the full citation of the original published article followed by the IEEE copyright line: © 20XX IEEE. If you are reusing a substantial portion of your article and you are not the senior author, obtain the senior author’s approval before reusing the text.

- **Graphics and tables:** The IEEE copyright line (© 20XX IEEE) should appear with each reprinted graphic and table.

- **Full text article:** Include the following copyright notice in the references: “© 20XX IEEE. Reprinted, with permission, from [full citation of original published article].”

Figure A.2 A screenshot granting permission from IEEE.