ML-descent:
an optimization algorithm for FWI using machine learning

Bingbing Sun and Tariq Alkhalifah

bingbing.sun@kaust.edu.sa, tariq.alkhalifah@kaust.edu.sa

Physical Sciences and Engineering

King Abdullah University of Science and Technology

Thuwal, 23955, Saudi Arabia

(February 8, 2021)

Running head:

ABSTRACT

Full-waveform inversion (FWI) is a nonlinear optimization problem, and a typical optimization algorithm such as nonlinear conjugate gradient or Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) would iteratively update the model mainly along the gradient-descent direction of the misfit function or a slight modification of it. Based on the concept of meta-learning, rather than using a hand-designed optimization algorithm, we train the machine (represented by a neural network) to learn an optimization algorithm, entitled "ML-descent" and apply it in FWI. Using a Recurrent Neural Network (RNN), we use the gradient of the misfit function as input, and the hidden states in the RNN incorporate the history information of the gradient similar to an LBFGS algorithm. However, unlike the fixed form of the LBFGS algorithm, the ML version evolves in response to the gradient. The loss function for training is formulated as a weighted summation of the L2-norm of the data residuals in the original inverse problem. As with any well-defined nonlinear in-

verse problem, the optimization can be locally approximated by a linear convex problem, and thus, to accelerate the training, we train the neural network by minimizing randomly generated quadratic functions instead of performing time-consuming FWIs. In order to further improve the accuracy and robustness, we use a Variational Autoencoder (VAE) that projects and represents the model in latent space. We use the Marmousi and the Overthrust examples to demonstrate that the ML-descent method shows faster convergence and outperforms conventional optimization algorithms. The energy in the deeper part of the models can be recovered by the ML-descent even when the pseudo-inverse of the Hessian is not incorporated in the FWI update.

INTRODUCTION

Full-waveform inversion (FWI) is a high-resolution subsurface imaging method. It utilizes the full information of the pre-stack seismic dataset, i.e., reflections, refractions, diving waves, P- and S-waves, surface waves, amplitudes, and phases. A successful implementation of FWI can invert for the subsurface properties such as P- and S-wave velocities, density, attenuation, and anisotropy parameters with high resolution in the limit of the Rayleigh criterion (Bleistein et al., 2013).

Mathematically, FWI can be considered as a nonlinear inverse problem. Although global optimization schemes such as the Monte Carlo method (Jin and Madariaga, 1994; Sambridge and Mosegaard, 2002), genetic algorithms (Sen and Stoffa, 1992; Jin and Madariaga, 1993), simulated annealing (Kirkpatrick et al., 1983; Datta and Sen, 2016) have shown potential in solving the FWI problem, those global optimization methodologies typically require an evaluation of the misfit function for tens of thousands of times. For real applications, especially in 3D, where we have hundreds of millions of unknown parameters, these strategies are impractical considering the current computational capabilities. Thus, current FWI methods still rely on local optimization schemes in which we mainly update the model along descent directions.

There are various iterative methods for solving the nonlinear FWI optimization problem, including the steepest descent method, nonlinear conjugate-gradient, and Newton's methods, like the Gauss-Newton method, and various quasi-Newton methods (Tarantola, 2005; Ma and Hale, 2012). Compared to the steepest-descent and nonlinear conjugate-gradient methods, Newton's methods generally converge faster in fewer iterations. However, for a model size of $\mathcal{O}(N)$, full Newton's methods require evaluating the inverse of the Hessian ma-

trix of size $\mathcal{O}(N^2)$, which is computationally prohibitive especially for 3D problems. Thus, various methodologies have been proposed to approximate the Hessian matrix. For example, the Gaussian-Newton method ignores the second-order term that accounts for double scattering in the model. Quasi-Newton methods do not compute the Hessian matrix, but instead, iteratively update a Hessian matrix approximation. Different strategies for such Hessian updating can be utilized, such as the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm (Fletcher, 1987) and the Davidon-Fletcher-Powell (DFP) algorithm (Davidon, 1991). However, although the BFGS method reduces the computation time required to approximate a Hessian matrix, it does not decrease the amount of memory required to store the approximated Hessian.

The limited-memory BFGS (LBFGS) (Liu and Nocedal, 1989) method can be considered as a practical implementation of the quasi-Newton method. It does not explicitly compute or store any type of Hessian matrix. Instead, LBFGS only stores information, such as model changes and gradient changes, for a limited number M of previous iterations. LBFGS then uses the stored information to implicitly form an inverse of the approximated Hessian and obtain an update direction for the next iteration.

All the aforementioned hand-crafted optimization algorithms are applicable to general inverse problems and are not specific to FWI. In this work, we propose an optimization algorithm for FWI based on the concept of meta-learning from the Machine-Learning (ML) community. Meta-learning (Schmidhuber, 1987; Vilalta and Drissi, 2002) is an automatic learning methodology in ML. It is flexible in solving learning problems and tries to improve the performance of existing learning algorithms or to learn (extract) the learning algorithms itself. It is also referred to as "learning to learn".

In this paper, we try to learn the optimization algorithm in FWI. Specifically, we take the gradient of the misfit function at the current step as input and output the updating direction using a Recurrent Neural Network (RNN). RNNs have wide applications in machine learning, such as in natural language understanding, language generation, video processing and many other tasks (Mandic and Chambers, 2001). Recently, Andrychowicz et al. (2016) propose to use an RNN for designing optimization algorithms. Their learned algorithms outperform generic, hand-crafted alternatives in tasks for which they are trained, and also generalize well to new tasks with a similar structure. Thus, the majority of the work presented in this paper is inspired by their work. The difference is that they applied the learned algorithms to problems popular in machine learning, such as classification, while we applied such a learned algorithm to a specific inverse problem in geoscience, i.e, FWI. Our main purpose is to determine whether the machine can learn how to better update the model in FWI, and thus, help in accelerating the convergence.

Another important aspect in improving the efficiency of optimization algorithms is related to re-parameterization (or regularization) of the model. For example, Loris et al. (2007) perform a high-contrasting tomographic inversion in the wavelet-domain with an l_1 norm constraint. Ma et al. (2012) reduce the parameters in FWI significantly by an image-guided projection of the model. Li et al. (2016) suggest a curvelet-domain FWI, and their method allows efficient updating of the model using a random subset of the full data. Zhu et al. (2017) introduce the Sparse Orthonormal Transform (SOT) learned from small patches in the model perturbation. Xue et al. (2017) improve the robustness of FWI by enforcing the sparsity of the solutions in the seislet domain (Fomel and Liu, 2010). All those approaches try to introduce prior information into the inversion by representing the model as a linear combination of basis functions. Those basis functions can be hand-crafted

such as wavelets, curvelets (Sun et al., 2009, 2011) or learned online (Zhu et al., 2017). In fact, these kinds of model representations can be addressed adaptively with a machine-learning algorithm, e.g., an autoencoder. By learning an optimal representation in an unsupervised (or self-supervised) manner, the autoencoder often represents the model in a reduced latent-space. Such latent space representation is nonlinear, multi-scale and adheres to the structure within the data used for training (Hinton and Salakhutdinov, 2006). In the seismic-exploration community, autoencoders are already used in various applications, such as interpolation (Mandelli et al., 2018), reservoir facies prediction (Qian et al., 2018), as well as low-wavenumber reconstruction in salt bodies (Gao et al., 2019). In this paper, we use a Variational Autoencoder (VAE) (Kingma and Welling, 2019). We will represent and optimize the model in the latent space combined with the machine-learned optimization algorithm.

We will first describe in detail how we can teach a machine to construct an optimization algorithm. Then, we introduce the Variational Autoencoder and describe a latent-space ML-descent procedure. At last, we compare our ML-descent with state-of-the-art optimization algorithms, and demonstrate the effectiveness and good performance of the ML-descent using well-known synthetic examples, such as the Marmousi and Overthrust models.

MACHINE LEARNING OF AN OPTIMIZATION ALGORITHM

In this section, we will first review the importance of utilizing the gradient history of an optimization problem in improving the model updating direction. We then utilize this concept in training a neural network algorithm to learn how to improve the update to satisfy the same optimization objective.

The history dependency feature of the optimization algorithms

In nonlinear inversions, a first-order optimization algorithm such as the nonlinear conjugate-gradient or LBFGS method can be summarized as:

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \gamma \Delta \mathbf{x}_{t-1}([\mathbf{g}]_{t-1}, [\Delta \mathbf{x}]_{t-2}), \quad (1)$$

where γ is the step length (learning rate), and at iteration t , $\mathbf{x}_t$ is the vector we want to invert for in the optimization problem. The vector $\Delta \mathbf{x}_{t-1}$ is the updating direction, which is a function of the gradient $[\mathbf{g}]_{t-1} = (\mathbf{g}_{t-k}, \mathbf{g}_{t-k+1}, \mathbf{g}_{t-k+2}, \dots, \mathbf{g}_{t-1})$ and the updating direction $[\Delta \mathbf{x}]_{t-2} = (\Delta \mathbf{x}_{t-k}, \Delta \mathbf{x}_{t-k+1}, \Delta \mathbf{x}_{t-k+2}, \dots, \Delta \mathbf{x}_{t-2})$ in the history, where k denotes how many history steps incorporated. The difference between various optimization algorithms is how to compute the current updating direction based on such history information, and the size of k .

For example, one kind of Nonlinear Conjugate Gradient (NCG) method suggests the following form for the updating direction (Dai and Yuan, 1999):

$$\Delta \mathbf{x}_{t-1} = -\mathbf{g}_{t-1} + \beta_{t-1} \Delta \mathbf{x}_{t-2}, \quad (2)$$

where,

$$\beta_{t-1} = \frac{\mathbf{g}_{t-1}^T \mathbf{g}_{t-1}}{\Delta \mathbf{x}_{t-2}^T (\mathbf{g}_{t-1} - \mathbf{g}_{t-2})}. \quad (3)$$

Here, the symbol 'T' denotes the vector transpose. We can see that for this kind of NCG algorithm, the updating direction will be determined by the gradient and updating directions of two iterations (current and previous, or $k=2$). While in LBFGS, the updating direction will depend on the gradient and updating information in k previous iterations. This history dependency feature in NCG and LBFGS demonstrates the importance of keeping track of

the gradient history. Thus, a machine learned optimization algorithms should aim to mimic this history utilization feature, and possibly improve on it as machine learned algorithms are data-adaptive and more adaptable.

Recurrent neural network (RNN)

As we discussed in the previous section, we need to keep track of the history information when we train optimization algorithms using Machine Learning. The recurrent neural network (RNN) is an ideal neural network architecture for this purpose. Figure 1a shows an RNN with an input vector $\mathbf{x}_t$ and an output vector $\mathbf{y}_t$. A loop allows the information to be passed from one step of the network to the next (one iteration of FWI to another). As shown in Figure 1b, these loops in the RNN can be thought of as multiple copies of the same network, each passing a message to a successor. This chain-like nature reveals that RNN is intimately related to time sequences and lists, and they are the natural architecture of neural networks to use for such an evolution problem.

There are variations of RNN, such as the Long Short-Term Memory (LSTM) (Gers et al., 2003) and the Gated Recurrent Unit (GRU) (Dey and Salemt, 2017). In this paper, we use the LSTM for designing a network to learn the optimization algorithm. LSTM is a special kind of RNN that avoids the problem of vanishing gradients and is capable of learning long-term dependencies (Refer to the appendix A for a review of the LSTM).

Thus, in our inverse problem, suppose we have a current model $\mathbf{x}_t$, the input to the LSTM is the gradient $\mathbf{g}_t$ together with the state variable $\mathbf{c}_t$ and the hidden state variable $\mathbf{h}_t$ at the current time step (iteration in the optimization). The output would be the updating direction $\Delta\mathbf{x}_t$ together with the state variable $\mathbf{c}_{t+1}$ and the hidden state variable $\mathbf{h}_{t+1}$ used

for the next time step:

$$(\Delta \mathbf{x}_t, \mathbf{c}_{t+1}, \mathbf{h}_{t+1}) = \text{LSTM}(\mathbf{g}_t, \mathbf{c}_t, \mathbf{h}_t; \theta), \quad (4)$$

where θ are the parameters of the LSTM that we need to train and optimize. From Figure 2, we can see that LSTM is used recursively. We have only one neural network of LSTM, and in each time step, they share the same parameter θ for training, i.e., we only have one set of parameters θ during the training over as many steps as needed. We unroll the network along the time direction for demonstration purposes only. The state and the hidden state variables, $\mathbf{c}_t$ and $\mathbf{h}_t$, which appear in both of the input and output of LSTM, keep track of the history of the updating information. Thus, they allow the resulting algorithm to use the history of the updating like in the LBFGS method. In fact, many recent learning procedures, such as the adaptive moment (ADAM) estimation, also utilize the history information by taking the momentum into consideration.

Loss function for the meta-learning

The loss function for training would be a weighted summation of the loss function in the original inverse problem:

$$L = \sum_t w_t f(\mathbf{x}_t(\theta)), \quad (5)$$

where f is the loss function for the original inverse problem (e.g., the least-square difference), and w_t is a weighting variable. Since the updating direction is determined by the output of RNN, f is a function of the parameter θ of the RNN and as a result, so is L . Specifically, the derivative of the loss function f with respect to the model $\mathbf{x}$ computes the gradient $\mathbf{g}$ for updating the model; while the derivative of the loss function L (the meta-learning loss) with respect to the RNN parameters θ provides the updates for the network. Also note

that, we have a weighted summation of the misfit function over all time steps (iterations). We can, for example, select only the final step of the misfit value as the meta-loss function (remove the summation in equation 5), but this makes the training less stable. When back-propagating the residual, the gradient reduces considerably in value when it propagates back to earlier time steps. Thus, in the examples shown here, we select the weighting w_t to be 1, i.e., we give the same weight for the misfit value in each step.

The coordinate-wise implementation

For a typical 3D FWI application, the number of model parameters can be in the scale of millions. Optimizing for this scale with a fully connected RNN is not feasible as it would require huge hidden states and an enormous number of parameters (note that the parameters of the RNN would be in the order of $\mathcal{O}(N^2)$, where N is the size of the input). To solve this problem, we use a modified RNN, which operates coordinate-wise on the parameters of the objective function, as shown in Figure 3. Different coordinates (each pixel in the model represents one coordinate) share the same RNN parameters for updating, while the unique behavior for each coordinate is achieved by using separate hidden state variables. Thus, this coordinate-wise network architecture can reduce the memory computation burden significantly (from order $\mathcal{O}(N^2)$ to $\mathcal{O}(1)$).

Training of the RNN

We can run FWI on randomly generated velocity models to train the RNN. However, a typical time-domain FWI requires solving a partial differential equation to model the data and to construct the gradient, which tends to be computationally expensive. However, any

nonlinear inversion problem can be approximated locally by a linear problem. Thus, we design various linear problems for efficient training. One simple option is to use a quadratic loss function f as:

$$f = ||\mathbf{W}\mathbf{x} - \mathbf{b}||_2^2. \quad (6)$$

We randomly choose the matrix $\mathbf{W}$ and vector $\mathbf{b}$ to formulate a particular inverse problem. Such training is far more efficient than using FWI as it avoids wave-equation modeling, and alternatively, we only need one matrix multiplication and one vector subtraction for the modeling. The weighting matrix is chosen to be 10×10 , and vector $\mathbf{b}$ is a ten-dimensional vector. Their elements are drawn randomly from a Gaussian distribution. Each function was optimized for 100 iterations and every 20 iterations, we evaluate the summation of the misfit as defined in equation 5 and back-propagate the residual to update the parameters of the RNN. We run 100 epochs and in each epoch, we randomly generated 2000 quadratic functions. In Figure 4, we show the errors over epochs in training the RNN. We can see that the loss is reducing, indicating that the RNN can learn a specific optimization algorithm for updating a model within an inverse problem. In Figure 5, we show the performance of different optimization algorithms on the randomly sampled ten-dimensional quadratic functions. We can see that the ML-descent algorithm by RNN shows faster convergence than other state-of-the-art algorithms such as ADAM (Kingma and Ba, 2014) and RM-Sprop (Ruder, 2016). This example demonstrates the advantage of a learned optimization algorithm over hand-crafted ones. Later, we will apply this ML-descent algorithm for FWI model updating.

LATENT SPACE REPRESENTATION BY VARIATIONAL AUTOENCODER

Although we can apply the ML-descent to FWI directly, we seek to project the model in the latent space of an autoencoder for a more robust updating. The reason is that the ML-descent is implemented point-wisely. Projection into the latent space allows the resulting ML-descent to take the space coherency information into consideration.

Figure 6a shows a typical autoencoder with one hidden layer. In this case, the autoencoder includes two main parts: an encoder $q_{\theta_q}(\mathbf{z}|\mathbf{x})$ that maps the input $\mathbf{x}$ into the code $\mathbf{z}$ in the latent space, and a decoder $p_{\theta_p}(\mathbf{z}|\mathbf{x})$ that maps the code $\mathbf{z}$ to a reconstruction of the original input $\tilde{\mathbf{x}}$. More advanced autoencoders have been proposed for improved stability and accuracy such as the Sparse Autoencoder (SAE) (Makhzani and Frey, 2013), the Denoising Autoencoder (DAE) (Vincent et al., 2010), the Contractive Autoencoder (CAE) (Rifai et al., 2011) and the Variational Autoencoder (VAE) (Kingma and Welling, 2019). As shown in Figure 6b, except for the encoder and decoder part, VAE has a sampling step in the latent space resulting in a latent space with improved continuity. The loss function for the VAE can be defined as:

$$L(\theta_p, \theta_q) = -E_{\mathbf{z} \sim q_{\theta_q}(\mathbf{z}|\mathbf{x})}[\log p_{\theta_p}(\mathbf{x}|\mathbf{z})] + \text{KL}(q_{\theta_q}(\mathbf{z}|\mathbf{x})||p(\mathbf{z})). \quad (7)$$

Here, θ_p and θ_q represents the NN parameters of the encoder and decoder, respectively. The term $\text{KL}(q_{\theta_q}(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$ is the Kullback-Leibler (KL) divergence between the latent space distribution $q_{\theta_q}(\mathbf{z}|\mathbf{x})$ and a prior distribution $p(\mathbf{z})$:

$$\text{KL}(q_{\theta_q}(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) = E_{\mathbf{z} \sim p(\mathbf{z})} [\log p(\mathbf{z}) - \log q_{\theta_q}(\mathbf{z}|\mathbf{x})], \quad (8)$$

where $p(\mathbf{z})$ is the prior distribution about the latent space code $\mathbf{z}$, chosen to be a normal distribution with zero mean and unit variance, i.e., $p(\mathbf{z}) \sim \mathcal{N}(0, 1)$. The first term in the

loss function of equation 7 is the reconstruction loss. This term encourages the decoder to learn to reconstruct the data. The second term is a regularizer which attempts to "keep the representations $\mathbf{z}$ sufficiently diverse".

As long as we trained such VAE, we will represent the model of FWI in the latent space, i.e., the velocity model $\mathbf{m}$ is computed as:

$$\mathbf{m} = p_{\theta_p}(\mathbf{z}_{\mathbf{m}}), \quad (9)$$

where $z_{\mathbf{m}} = q_{\theta_q}(\mathbf{m})$ is the latent space code of the model. The gradient of the loss function with respect to the latent space $\mathbf{z}_{\mathbf{m}}$ can be computed using the chain rule:

$$\frac{\partial L}{\partial \mathbf{z}_{\mathbf{m}}} = \frac{\partial \mathbf{m}}{\partial \mathbf{z}_{\mathbf{m}}} \frac{\partial L}{\partial \mathbf{m}}. \quad (10)$$

In VAE, the smooth transition in the latent space represents a smooth transition in the original Euclidean space as well. This means that the Jacobian ($\frac{\partial \mathbf{m}}{\partial \mathbf{z}_{\mathbf{m}}}$ in equation 10) of the output of the decoder with respect to the input $\mathbf{z}$ is well defined. This is preferred for a more robust gradient projection in FWI based on equation 10.

We use the Marmousi model shown in Figure 10a for training the VAE. We decompose the model into small patches (6×6 pixels and grid size is 40 m), and then flip the resulting patches left to right, up to down to create more data. Finally, we use 40000 patches and normalize them to have zero mean and unit standard deviation for efficient training:

$$\hat{D} = \frac{(D - \mu_D)}{\sigma_D}, \quad (11)$$

where $\hat{D}$ is the dataset after normalization, D is the original dataset, and μ_D and σ_D are the mean and standard deviation for the original dataset, respectively. The neural network architecture for the VAE is shown in Figure 7. The VAE has two convolution layers with kernel sizes four and two, and then map to mean and variance with direct connects. We

train the VAE with latent space sizes nine and 18. In training, we randomly select 90% of the dataset for training and 10% for testing. We run the training for 10000 epochs with a relatively low learning rate ($lr = 3E-5$). Figure 8 shows the loss function over patches. We can see that the normalized loss value for the training and testing dataset has reduced to a small number (around 2%), indicating reasonable convergence of the VAE training. A larger latent space size leads to lower residuals, which is reasonable. Figure 9 shows the selections of the input dataset and its reconstruction for training and testing. The reasonably accurate reconstruction of the input dataset further demonstrates the accuracy of the trained VAE.

EXAMPLES

In this section, we apply our learned optimization algorithm, i.e., the ML-descent, to invert for a part of the Marmousi model. The true velocity v_{true} shown in Figure 10a extends 2 km in depth and 8 km laterally. As shown in Figure 10b, we create the initial model using a Gaussian smoothing of the true velocity model with a standard deviation of 200 m in both vertical and horizontal directions. The dataset is modeled using ten shots with a source interval of 800 m and 200 receivers with an interval of 40 m. The model grid size for computation is 40 m by 40 m (which is the same as that used for training the VAE). The source wavelet is a Ricker wavelet with a 5 Hz peak frequency. We perform a full-band dataset inversion and update the model using 100 iterations. To evaluate the performance of the optimization algorithm, we do not include the second-order information, e.g., the pseudo-Hessian, in the gradient computation. Figures 11 and 12 show the inversion result for different optimization methods. For a fair comparison, all the optimizations are implemented in the VAE latent space domain. We randomly search for the optimal learning rate for different optimization methods: Specially, for each algorithm, we define a

distribution for the learning rate, e.g., a log distribution and then randomly sample from the distribution to perform the training and the sample that gives the best result (minimum loss) will be selected. The learning rate is fixed during the optimization except for the LBFGS, which performs a line search. We use ten steps of memory storage for the LBFGS algorithm. In Figure 11, the size of the latent space is nine. Compared to the true model, we can see that the ML-descent (Figure 11f) and the RMSprop (Figure 11a) produce the best results with respect to accuracy and resolution. ML-descent is slightly better in the deeper part of the model. It is clear that without the pseudo-Hessian to compensate for illumination and geometric spreading effects, the stochastic gradient descent (SGD) and the nesterov accelerated gradient (NAG) method can hardly recover the model while the ADAM approach produces relatively strong artifacts at the boundary compared to other methods. Figure 11e shows the popular LBFGS result, which is acceptable, but it is low resolution compared to the ML-descent method, especially in the deeper part of the model. For comparison, in Figure 12, we show the result for a latent space size of 18 (double what we had before). The conclusion is the same, ML-descent yields the best results, closely followed by the RMSprop method. Figures 13a and 13b show the residuals over iterations for the latent space size nine and 18, respectively. As we observed before, the ML-descent method yields minimum residual (around 3% for both cases). Both RMSprop and ML-descent converges faster than the LBFGS method, and this observation is consistent with the conclusion by Sun et al. (2020). In Figure 17 of their paper, ADAM converges faster than LBFGS for their 2D Marmousi model. The second-order momentum used in the ADAM or RMSprop algorithms provides us with an adaptive learning rate; such adaptive learning rate for each pixel, can better compensate for the illumination and improve the resolution, resulting in faster convergence (Refer to the appendix B for a review of the

optimization algorithm in machine learning). The ML-descent is supposed to learn such adaptive learning rate features after training. We observe that the size of the latent space do not affect the ML-descent method. Conventional optimization methods, such as ADAM and RMSprop, prefer larger latent space, which results in lower residuals. To highlight the role of the latent space, we perform the optimizations in the original model space and show the inverted results in Figure 14 and the corresponding residual over iterations in Figure 15. Due to the coordinate-wise implementation, the ML-descent method (Figure 14f) in the original model space is rather noisy compared to the SGD, NAG, and LBFGS. In Figure 15, the LBFGS method in the original model space has the smallest residual. However, this residual is still larger than the latent space ML-descent method (Figure 12). This indicates that the combination of the coordinate-wise RNN and latent space implementation helped to reduce the noise and lead to a robust and fast implementation of the machine-learned optimization algorithm.

As the VAE is trained using the Marmousi model, we may suspect that the good performance of the latent space ML-descent algorithm for the Marmousi model is due to the bias of the VAE trained on the same model. Thus, we evaluate the different optimization algorithms using the already trained VAE by inverting for a different model, the 2D overthrust model. The true and initial models are shown in Figure 16. The size of the model is about 3.6 km in depth and 14 km horizontally. The initial model is obtained by smoothing the true model with a Gaussian smoother of a standard deviation of 400 m in both horizontal and vertical directions. The grid size for computation is 40 m by 40 m. We use a 5 Hz Ricker wavelet, and we place 20 shots, and 350 receivers evenly spread at the surface. As before, we randomly search for the optimal learning rate for different optimization methods. We run 100 iterations of FWI, and Figure 17 shows the inversion results using the different

algorithms with a latent space size of 18. Figure 18 shows the residual over iterations for different methods. From those results, we can easily identify that the ML-descent gives the best result (highest resolution and smallest residual). This indicates that although the VAE is trained on the Marmousi model, the resulting latent-space ML-descent can be applied to invert for other models as well. This feature can be attributed to the similarity in the small scale of patches between the Marmousi and the Overthrust model, and this would also apply to the inversion of other models.

Transfer learning: learn ML-descent algorithm by FWI

As we mentioned earlier, the main component of the ML-descent algorithm, i.e., the parameters of the RNN, are trained by inverting quadratic functions for efficient training. We then use the trained RNN in FWI applications directly, and as shown in previous examples, it has an advantage over hand-crafted optimization algorithms. In this section, we use the RNN trained with quadratic functions as an initial neural network model and then further train it using FWI. In the machine-learning community, such training is called transfer learning (Tan et al., 2018). We train the neural network for one task (the quadratic loss), which is similar to the target task (FWI). For training the RNN using FWI, we use the latent-space ML-descent with a latent space size of 18. We run FWI using the Marmousi model described before and in order to introduce randomness in the training, we generate the initial model by Gaussian smoothing of the true model with different (random) standard deviations: Specifically, the standard deviation is randomly chosen between 20 m and 200 m from a uniform distribution. The smoothing parameter in this range can guarantee that the inversion converges without facing the cycle-skipping issue. We run 100 iterations for one pass of FWI (one epoch). After each epoch, we run the inversion using the same initial

model shown in Figure 10a and keep track of the summation of the residual over iterations. Figure 19 shows such residuals over epochs, and it can be regarded as the testing residual over epochs. We can see that the training is converging, and that the loss drops about 30% after about 1000 epochs (note that the original RNN is already pre-trained by inverting quadratic functions and the initial loss is already small). The training process using FWI can be considered as adjusting the latent space ML-descent to accommodate the features of the gradient in FWI.

In Figure 20, we add the loss curve for the ML-descent with transfer learning to Figures 13b and 15. For the Marmousi model (Figure 20a), after transfer learning, the residual drops further, from around 10^{-3} to 10^{-5} ; almost two orders of magnitude while also dropping faster (note that in first 20 iterations, the residual drops below 10^{-3}). We expected to see such larger and faster residual drops because the transfer learning is applied to the Marmousi model. However, for the overthrust model (Figure 20b), the ML-descent after transfer learning still gives smaller residuals (the residual drops from around 10^{-3} to around 10^{-4}) and shows fast convergence (see the first 40 iterations). We also show the evolution of the overthrust model over iterations by the ML-descent with (Figure 21) and without transfer-learning (Figure 22). The fast convergence of the ML-descent with transfer learning is further demonstrated with those model evolutions.

Optimization with noisy record

In this section, we present the result of the overthrust model in which we add Gaussian random noise to the data. We adopt the following signal-to-noise ratio (S/N) definition:

$$S/N = 10 \log_{10} \left(\frac{\|s_{\text{true}}\|_2^2}{\|s_{\text{true}} - s\|_2^2} \right), \quad (12)$$

where s and s_{true} are the traces with and without noise, respectively. In this section, we use the RNN neural network after transfer learning for testing. The setup for the overthrust model is the same as above. Figure 23 shows one typical shot record with different levels of additive Gaussian random noise. Specially, the S/N for the record in Figure 23b is 5 dB, representing strong noise while the noise in Figure 23c is minor, leading to a high S/N of 20 dB. Figures 24 and 25 show the optimization results for those noisy records. Compared to the noise-free inversion result of Figures 21, the results with strong noise show some expected artifacts, and the result by the ML-descent shows high resolution compared to other algorithms in both cases. To verify the accuracy of the inverted model, we define the model misfit as the normalized L2-norm of the difference between the true m_{true} and inverted model m :

$$\text{Misfit}_{\text{model}} = \left\| \frac{m_{\text{true}} - m}{m_{\text{true}}} \right\|_2^2. \quad (13)$$

In Figure 26, we shows the model misfit curve over iterations. The fast convergence and small model residuals at the end further verify the robustness and good performance of the proposed ML-descent method.

Acoustic FWI with elastic dataset

To further verify the robustness of the proposed method, we perform an acoustic FWI using an elastic dataset modeled from the Marmousi-II model (Martin et al., 2006). As shown in Figure 27, the model extends about 16 km horizontally and 3.4 km in depth. The elastic dataset is modeled with a 5 Hz Ricker wavelet, and with 50 shots and 425 receivers spread evenly on the surface. We put both the sources and receivers at a depth of 20 m. We record the vertical particle velocity to represent the measured data. In Figure 28, we show a shot record. As shown in Figure 27d, the initial velocity for the acoustic FWI is given

by smoothing the true P-wave velocity using a Gaussian filter with a standard deviation of 400 m in both vertical and horizontal directions.

We run the inversion over 20 iterations. Figure 29 shows the inverted P-wave velocity. We can see that the ML-descent still shows high-resolution features, demonstrating its robustness in inverting the elastic dataset using an acoustic FWI engine. The convergence curves for the model misfit shown in Figure 30 further demonstrates the good performance of the ML-descent with minimum model residual.

CONCLUSION

We developed an optimization algorithm (ML-descent) learned using a machine. RNN is used to learn an optimization algorithm, which can use the history of the update, like LBFGS, to improve the convergence. A coordinate-wise implementation of the RNN is evaluated for efficient training of the network. To introduce regularization (smoothing) and improve the robustness of the ML-descent algorithm, we train a variational autoencoder and perform the optimization by projecting the model into a latent space. We train the RNN initially using randomly generated quadratic functions and then further train on FWI based on the concept of transfer learning. We use Marmousi and Overthrust models to demonstrate the good performance of the algorithm learned by the machine. An application of the proposed method in inverting an elastic dataset using acoustic engine further verified the potential of the ML-descent of being a practical approach for FWI optimization.

ACKNOWLEDGMENTS

We thank Editor-in-Chief Jeffrey Shragge for improving the manuscript. We appreciate Assistant Editor Antoine Guitton and Associate Editor Sjoerd A. L. de Ridder for their technical comments and suggestions that greatly improved the manuscript.

APPENDIX A

THE LONG SHORT-TERM MEMORY (LSTM) NETWORKS

In this appendix, we provide a brief introduction of the LSTM network. Figure A-1 shows the basic unit of an LSTM cell. We can see that LSTM takes the input x_t and keeps track of the cell state variable c_t and the hidden state variable h_{t-1} . The output of the LSTM is evaluated from the hidden state variable h_t . The key to LSTM is the cell state, the horizontal line running through the top of the diagram.

Another important concept is the gate, which is defined to optionally let information through. They are composed of a sigmoid activation function and a point-wise multiplication operation. By using gates, the LSTM has the ability to add or remove information corresponding to the cell state.

The first step in LSTM is to decide what information to throw away from the cell state using the forget-gate layer. It takes h_{t-1} and x_t and outputs a number between 0 and 1 using a sigmoid function:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \quad (\text{A-1})$$

where W_f and b_f are the weight and bias for the forget-gate, σ is sigmoid activation function. The next step is to decide what new information we will store in the cell state. A $\tanh$ activation function takes the vector $\tilde{C}_t$ as input and the input-gate layer decides which values we will update:

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C), \quad (\text{A-2})$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \quad (\text{A-3})$$

where W_C and W_i are the weights, and b_C and b_i are the biases. Thus, the new cell state can be computed as:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t. \quad (\text{A-4})$$

Finally, we need to decide what we are going to output. The output will be based on our cell state, but will be a filtered version using an output gate o_t :

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (\text{A-5})$$

$$h_t = o_t * \tanh(C_t), \quad (\text{A-6})$$

where W_o and b_o are the weight and bias for the output-gate, respectively.

APPENDIX B

OPTIMIZATION ALGORITHMS IN MACHINE LEARNING

In this appendix, we provide a brief review of the optimization algorithms used in machine learning. For detailed information regarding those algorithms, we refer you to Sun et al. (2019).

Considering a supervised learning problem, sample i of the training dataset corresponds

to the input of x_i with label y_i and the loss function is defined as:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^N \frac{1}{2} [y^i - \Phi(x^i; \theta)]^2, \quad (\text{B-1})$$

where N is the batch size and Φ denotes the neural network with parameter θ .

The **Stochastic Gradient Descent** (SGD) randomly samples N points from the dataset and updates the parameter θ using the gradient information only:

$$\theta_{t+1} = \theta_t - \eta g_t, \quad (\text{B-2})$$

where η is the learning rate, and θ_{t+1} and θ_t represents the neural network parameter at iteration $t + 1$ and t , respectively. g_t is the gradient of the loss $\mathcal{L}(\theta)$ with respect to θ :

$$g_t = \frac{\partial \mathcal{L}(\theta_t)}{\partial \theta}. \quad (\text{B-3})$$

SGD is prone to being trapped in a local minimum and one way to improve it is to use **momentum** to preserve the influence of the previous update direction on the next iteration to a certain degree. The updating direction for the moment method is modified as :

$$v_t = -\eta g_t + \alpha v_{t-1}, \quad (\text{B-4})$$

where α is the momentum weighting parameter and v_t and v_{t-1} denote current and previous updating directions, respectively.

The **Nesterov Accelerated Gradient descent** (NAG) further improves the moment method. NAG uses the gradient of the future position instead of current position for updating. The detailed updating formulas are as follows:

$$\begin{cases} \tilde{\theta}_{t+1} = \theta_t + \alpha v_{t-1} \\ v_t = \alpha v_{t-1} - \eta \frac{\partial \mathcal{L}(\tilde{\theta}_{t+1})}{\partial \theta} \\ \theta_{t+1} = \theta_t + v_t \end{cases} \quad (\text{B-5})$$

The **AdaGrad** method is proposed to adjust the learning rate η dynamically based on the historical gradient:

$$\begin{cases} g_t = \frac{\partial \mathcal{L}(\theta_t)}{\partial \theta} \\ V_t = \sqrt{\sum_{i=1}^{i=t} g_i^2} + \epsilon \\ \theta_{t+1} = \theta_t - \eta \frac{g_t}{V_t} \end{cases}, \quad (\text{B-6})$$

where $V(t)$ is referred to as the second-order cumulative momentum and ϵ is a small number to avoid division over zero.

In AdaGrad, as the training time increases, the accumulated gradient V_t becomes larger and larger, making the learning rate approach zero. **AdaDelta** and **RMSprop** were further proposed to overcome this issue by focusing on the gradients in a window over period, and an exponential moving average is used to calculate the second-order cumulative momentum:

$$V_t = \sqrt{\beta V_{t-1} + (1 - \beta) g_t^2}, \quad (\text{B-7})$$

where β is the exponential decay parameter.

Adaptive moment estimation (ADAM) utilizes both moment and second-order moment to achieve fast convergence:

$$\begin{cases} m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ V_t = \sqrt{\beta_2 V_{t-1} + (1 - \beta_2) g_t^2} \\ \theta_{t+1} = \theta_{t-1} - \eta \frac{\sqrt{1 - \beta_2}}{1 - \beta_1} \frac{m_t}{V_t + \epsilon} \end{cases}, \quad (\text{B-8})$$

where β_1 and β_2 are the exponential decay rates.

REFERENCES

- Andrychowicz, M., M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas, 2016, Learning to learn by gradient descent by gradient descent: arXiv:1606.04474.
- Bleistein, N., J. K. Cohen, and W. John Jr, 2013, Mathematics of multidimensional seismic imaging, migration, and inversion: Springer Science & Business Media.
- Dai, Y., and Y. Yuan, 1999, A nonlinear conjugate gradient method with a strong global convergence property: SIAM Journal on Optimization, **10**, no. 1, 177–182.
- Datta, D., and M. K. Sen, 2016, Estimating a starting model for full-waveform inversion using a global optimization method: Geophysics, **81**, no. 4, R211–R223.
- Davidon, W., 1991, Variable metric method for minimization: SIAM Journal on Optimization, **1**, no. 1, 1–17.
- Dey, R., and F. M. Salemt, 2017, Gate-variants of gated recurrent unit (gru) neural networks: 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), 1597–1600.
- Fletcher, R., 1987, Practical methods of optimization: John Wiley Sons.
- Fomel, S., and Y. Liu, 2010, Seislet transform and seislet frame: Geophysics, **75**, no. 3, V25–V38.
- Gao, Z., Z. Pan, J. Gao, and Z. Xu, 2019, Building long-wavelength velocity for salt structure using stochastic full waveform inversion with deep autoencoder based model reduction: SEG Technical Program Expanded Abstracts, 1680–1684.
- Gers, F. A., N. N. Schraudolph, and J. Schmidhuber, 2003, Learning precise timing with LSTM recurrent networks: Journal of Machine Learning Research, **3**, no. 29, 115–143.
- Hinton, G. E., and R. R. Salakhutdinov, 2006, Reducing the dimensionality of data with neural networks: Science, **313**, no. 5786, 504–507.

- Jin, S., and R. Madariaga, 1993, Background velocity inversion with a genetic algorithm: *Geophysical Research Letters*, **20**, no. 2, 93–96.
- , 1994, Nonlinear velocity inversion by a two step monte carlo method.: *Geophysics*, **59**, no. 4, 577–590.
- Kingma, D., and J. Ba, 2014, Adam: A method for stochastic optimization: [arXiv:1412.6980](https://arxiv.org/abs/1412.6980).
- Kingma, D., and M. Welling, 2019, Auto-encoding variational Bayes: [arXiv:1312.6114](https://arxiv.org/abs/1312.6114), 1–14.
- Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi, 1983, Optimization by simulated annealing: *Science*, **220**, no. 4598, 671–680.
- Li, X., E. Esser, and F. J. Herrmann, 2016, Modified Gauss-Newton full-waveform inversion explained why sparsity-promoting updates do matter: *Geophysics*, **81**, no. 3, R125–R138.
- Liu, D. C., and J. Nocedal, 1989, On the limited memory BFGS method for large scale optimization: *Mathematical Programming*, **45**, no. 1, 503–528.
- Loris, I., G. Nolet, I. Daubechies, and F. A. Dahlen, 2007, Tomographic inversion using l1-norm regularization of wavelet coefficients: *Geophysical Journal International*, **170**, no. 1, 359–370.
- Ma, Y., and D. Hale, 2012, Quasi-Newton full-waveform inversion with a projected Hessian matrix: *Geophysics*, **77**, no. 5, R207–R216.
- Ma, Y., D. Hale, B. Gong, and Z. J. Meng, 2012, Image-guided sparse-model full waveform inversion: *Geophysics*, **77**, no. 4, R189–R198.
- Makhzani, A., and B. Frey, 2013, K-sparse autoencoders: [arXiv:1312.5663](https://arxiv.org/abs/1312.5663).
- Mandelli, S., F. Borra, V. Lipari, P. Bestagini, A. Sarti, and S. Tubaro, 2018, Seismic data interpolation through convolutional autoencoder: *SEG Technical Program Expanded Ab-*

- stracts, 4101–4105.
- Mandic, D. P., and J. Chambers, 2001, Recurrent neural networks for prediction: Learning algorithms, architectures and stability: John Wiley & Sons, Inc.
- Mani, K., 2019, GRU’s and LSTM’s: https://miro.medium.com/max/796/1*-kdbdyzr7lpimgb3airkow.png. (accessed Jan 12, 2020).
- Martin, G. S., R. Wiley, and K. J. Marfurt, 2006, Marmousi2: An elastic upgrade for marmousi: *The Leading Edge*, **25**, no. 2, 156–166.
- Qian, F., M. Yin, X.-Y. Liu, Y.-J. Wang, C. Lu, and G.-M. Hu, 2018, Unsupervised seismic facies analysis via deep convolutional autoencoders: *Geophysics*, **83**, no. 3, A39–A43.
- Rifai, S., P. Vincent, X. Muller, X. Glorot, and B. Y., 2011, Contractive auto-encoders: explicit invariance during feature extraction: *Proceedings of the 28th International Conference on Machine Learning*.
- Ruder, S., 2016, An overview of gradient descent optimization algorithms: [arxiv.org](https://arxiv.org/abs/1609.04747).
- Sambridge, M., and K. Mosegaard, 2002, Monte carlo methods in geophysical inverse problems.: *Reviews of Geophysics*, **40**, no. 3, 1009.
- Schmidhuber, J., 1987, Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook: Diploma thesis, Technische Universitat Munchen, Germany.
- Sen, M. K., and P. L. Stoffa, 1992, Rapid sampling of model space using genetic algorithms: examples from seismic waveform inversion: *Geophysical Journal International*, **108**, no. 1, 281–292.
- Sun, B., H. Chauris, and J. Ma, 2011, 3D post-stack one-way migration using curvelets: *Journal of Seismic Exploration*, **20**, no. 3, 257.
- Sun, B., J. Ma, H. Chauris, and H. Yang, 2009, Solving wave equations in the curvelet

- domain: A multi-scale and multi-directional approach: *Journal of Seismic Exploration*, **18**, no. 4, 385–399.
- Sun, J., Z. Niu, K. A. Innanen, J. Li, and D. O. Trad, 2020, A theory-guided deep learning formulation and optimization of seismic waveform inversion: *Geophysics*, **85**, no. 2, R87–R89.
- Sun, S., Z. Cao, H. Zhu, and J. Zhao, 2019, A survey of optimization methods from a machine learning perspective: *IEEE Transactions on Cybernetics*, 1–14.
- Tan, C., F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, 2018, A survey on deep transfer learning: *arXiv:1808.01974*.
- Tarantola, A., 2005, *Inverse problem theory and methods for model parameter estimation*: Society for Industrial and Applied Mathematics.
- Vilalta, R., and Y. Drissi, 2002, A perspective view and survey of meta-learning: *Artificial Intelligence Review*, **18**, no. 2, 77–95.
- Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, 2010, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion: *Journal of Machine Learning Research*, **11**, no. 38, 3371–3408.
- Xue, Z., H. Zhu, and S. Fomel, 2017, Full-waveform inversion using seislet regularization: *Geophysics*, **82**, no. 5, A43–A49.
- Zhu, L., E. Liu, and J. H. McClellan, 2017, Sparse-promoting full-waveform inversion based on online orthonormal dictionary learning: *Geophysics*, **82**, no. 2, R87–R107.

LIST OF FIGURES

- 1 A diagram of (a) the RNN and its (b) unrolled version.

- 2 A diagram of the LSTM network: Input is the gradient $\mathbf{g}$ and output is the updating direction $\Delta\mathbf{x}$. The state and the hidden state variables, i.e., $\mathbf{c}$ and $\mathbf{h}$ are used to keep track of and utilize the history information: $(\mathbf{d}_t, \mathbf{c}_{t+1}, \mathbf{h}_{t+1}) = \text{LSTM}(\mathbf{g}_t, \mathbf{c}_t, \mathbf{h}_t; \theta)$, where θ represents the parameters for LSTM, which will be updated during the training.

- 3 A diagram of one step of coordinate-wise RNN updating. Here x_n is the coordinate-wise model (one pixel) representation while g_n is the associated coordinate-wise gradient. All RNNs have the shared parameters θ , but separate state variables.

- 4 The loss function for the ML-descent training over epochs. In each epoch, we randomly generate 100 ten-dimensional quadratic functions for training.

- 5 The performance of different optimization algorithms on randomly sampled 10-dimensional quadratic functions. The ML-descent, which is a learned optimization algorithm using RNN, shows faster convergence than other hand-crafted algorithms. Note the results is averaged over 100 randomly generated quadratic functions.

- 6 A diagram of the neural network for typical (a) autoencoder and (b) Variational autoencoder (VAE).

- 7 The neural network architecture for the VAE: We set the size of the latent space to be 18. The input size is six pixel by six pixel patches. The encoder consists of two convolution layers and the convolution kernel size is four by four and two by two, respectively. We have one direct connect layer, which maps the output of the convolution larger to the mean μ and variance σ^2 , both of them has size 18. We sample from a Gaussian distribution $\mathcal{N}(\mu, \sigma^2 \mathbf{I})$ to get the latent space code $\mathbf{z}$. The decoder is similar to the encoder, but in a transpose manner.

8 The training of the VAE: the size of the latent space is (a) nine and (b) 18. In the training, we use an ADAM optimization and set a relatively small learning rate. For both cases, the residual drops relatively fast.

9 For VAE, and a latent space size of nine: (a) training dataset, (b) training dataset reconstruction, (c) testing dataset, (d) testing dataset reconstruction. For a latent space size of 18: (e) training dataset, (f) training dataset reconstruction, (g) testing dataset, (h) testing dataset reconstruction.

10 (a) The true and (b) initial Marmousi velocity model.

11 The inverted velocity for the Marmousi model using (a) the RMSprop, (b) the ADAM, (c) the SGD, (d) the NAG, (e) the L-BFGS and, (f) the ML-descent methods. The update for the optimization is performed in the latent space domain and the size of the latent space is nine.

12 The inverted velocity for the Marmousi model using, (a) the RMSprop, (b) the ADAM, (c) the SGD, (d) the NAG, (e) the L-BFGS and, (f) the ML-descent methods. The update for the optimization is performed in the latent space domain and the size of the latent space is 18.

13 The residual over iterations for different optimization methods in the latent space: The size of the latent space is (a) nine and (b) 18.

14 The inverted velocity for the Marmousi model using (a) the RMSprop, (b) the ADAM, (c) the SGD, (d) the NAG, (e) the LBFGS and, (f) the ML-descent methods. The update for the optimization is performed in the original model space not the VAE latent space.

15 The residuals over iterations for the inversion of the Marmousi model in the original model space for the various update schemes.

- 16 (a) The true and, (b) initial Overthrust velocity model.
- 17 The inverted velocity for the Overthrust model using (a) the RMSprop, (b) the ADAM, (c) the SGD, (d) the NAG, (e) the L-BFGS and (f) the ML-descent methods. The update for the optimization is performed in the latent space domain and the size of the latent space is 18.
- 18 The residuals over iterations for the inversion of the Overthrust model for the various update schemes.
- 19 The residual over epochs for training the ML-descent by inverting for the Marmousi model.
- 20 The residuals over iterations for the various optimization methods with the latent space size of 18. (a) Marmousi model and (b) Overthrust model. Note that, for curve labelled with ML-descent, the RNN is trained with the quadratic functions only. The curve labeled with ML-descent (transfer) uses the pertained RNN from ML-descent and apply further training using FWI (of the Marmousi model).
- 21 The latent space ML-descent optimization of the overthrust model. The inverted model at iterations (a) 10, (b) 15, (c) 20, (d) 25, (e) 30, (f) 40, (g) 50 and (h) 90. Note that the ML-descent algorithm is trained using FWI on the Marmousi model with transfer learning.
- 22 The latent space ML-descent optimization of the overthrust model. The inverted model at iterations (a) 10, (b) 15, (c) 20, (d) 25, (e) 30, (f) 40, (g) 50 and (h) 90. Note the ML-descent algorithm here is trained using quadratic functions only.
- 23 A typical shot record for the overthrust model. (a) noise free shot record, shot record added with Gaussian random noise for (b) S/N 5 dB and (c) S/N 20 dB.
- 24 The inverted velocity for the Overthrust model using (a) the RMSprop, (b) the

ADAM, (c) the SGD, (d) the NAG, (e) the L-BFGS and (f) the ML-descent methods. The update for the optimization is performed in the latent space domain and the size of the latent space is 18. The RNN used is that after transfer learning. The S/N of the record for inversion is 5 dB as that shown in Figure 23b.

25 The inverted Overthrust velocity models using (a) the RMSprop, (b) the ADAM, (c) the SGD, (d) the NAG, e) the L-BFGS and (f) the ML-descent methods. The update for the optimization is performed in the latent space domain with latent space size of 18. We use the RNN trained by the transfer learning. The S/N of the shot record used for the inversion is 20 dB as shown in Figure 23c.

26 The model residuals of the overthrust model over iterations for the various optimization methods in the latent space. The size of the latent space is 18. The curves corresponding to inversion using record of (a) S/N 5 dB, (b) S/N 20 dB. We use the RNN trained by the transfer learning.

27 (a) P-wave velocity, (b) S-wave velocity, (c) density for the Marmousi-II elastic model and (d) Initial P-wave velocity model for the acoustic FWI. It is a smoothed version of the P-wave velocity model shown in (a).

28 A typical shot record for the elastic Marmousi-II model.

29 The inverted velocity for the elastic Marmousi-II model using (a) the RMSprop, (b) the ADAM, (c) the SGD, (d) the NAG, (e) the L-BFGS and (f) the ML-descent methods. The update for the optimization is performed in the latent space domain and the size of the latent space is 18. We use the RNN trained by the transfer learning.

30 The model residuals of the Marmousi-II model over iterations for the various optimization methods in the latent space: The size of the latent space is 18. The RNN used is that after transfer learning.

A-1 The basic unit of a LSTM cell (Mani, 2019). C_t and h_t are the state variables. x_t is the input. f_t , i_t and o_t corresponds to the forget gate, input gate and output gate, respectively. Refer to appendix A for more information.

Figure 1: A diagram of (a) the RNN and its (b) unrolled version.

Figure 2: A diagram of the LSTM network: Input is the gradient $\mathbf{g}$ and output is the updating direction $\Delta \mathbf{x}$. The state and the hidden state variables, i.e., $\mathbf{c}$ and $\mathbf{h}$ are used to keep track of and utilize the history information: $(\mathbf{d}_t, \mathbf{c}_{t+1}, \mathbf{h}_{t+1}) = \text{LSTM}(\mathbf{g}_t, \mathbf{c}_t, \mathbf{h}_t; \theta)$, where θ represents the parameters for LSTM, which will be updated during the training.

Figure 3: A diagram of one step of coordinate-wise RNN updating. Here x_n is the coordinate-wise model (one pixel) representation while g_n is the associated coordinate-wise gradient. All RNNs have the shared parameters θ , but separate state variables.

Figure 4: The loss function for the ML-descent training over epochs. In each epoch, we randomly generate 100 ten-dimensional quadratic functions for training.

—

Figure 5: The performance of different optimization algorithms on randomly sampled 10-dimensional quadratic functions. The ML-descent, which is a learned optimization algorithm using RNN, shows faster convergence than other hand-crafted algorithms. Note the results is averaged over 100 randomly generated quadratic functions.

Figure 6: A diagram of the neural network for typical (a) autoencoder and (b) Variational autoencoder (VAE).

Figure 7: The neural network architecture for the VAE: We set the size of the latent space to be 18. The input size is six pixel by six pixel patches. The encoder consists of two convolution layers and the convolution kernel size is four by four and two by two, respectively. We have one direct connect layer, which maps the output of the convolution larger to the mean μ and variance σ^2 , both of them has size 18. We sample from a Gaussian distribution $\mathcal{N}(\mu, \sigma^2 \mathbf{I})$ to get the latent space code $\mathbf{z}$. The decoder is similar to the encoder, but in a transpose manner.

(a)

(b)

Figure 8: The training of the VAE: the size of the latent space is (a) nine and (b) 18. In the training, we use an ADAM optimization and set a relatively small learning rate. For both cases, the residual drops relatively fast.

Figure 9: For VAE, and a latent space size of nine: (a) training dataset, (b) training dataset reconstruction, (c) testing dataset, (d) testing dataset reconstruction. For a latent space size of 18: (e) training dataset, (f) training dataset reconstruction, (g) testing dataset, (h) testing dataset reconstruction.

—

Figure 10: (a) The true and (b) initial Marmousi velocity model.

Figure 11: The inverted velocity for the Marmousi model using (a) the RMSprop, (b) the ADAM, (c) the SGD, (d) the NAG, (e) the L-BFGS and, (f) the ML-descent methods. The update for the optimization is performed in the latent space domain and the size of the latent space is nine.

Figure 12: The inverted velocity for the Marmousi model using, (a) the RMSprop, (b) the ADAM, (c) the SGD, (d) the NAG, (e) the L-BFGS and, (f) the ML-descent methods. The update for the optimization is performed in the latent space domain and the size of the latent space is 18.

(a)

(b)

Figure 13: The residual over iterations for different optimization methods in the latent space: The size of the latent space is (a) nine and (b) 18.

Figure 14: The inverted velocity for the Marmousi model using (a) the RMSprop, (b) the ADAM, (c) the SGD, (d) the NAG, (e) the LBFGS and, (f) the ML-descent methods. The update for the optimization is performed in the original model space not the VAE latent space.

Figure 15: The residuals over iterations for the inversion of the Marmousi model in the original model space for the various update schemes.

Figure 16: (a) The true and, (b) initial Overthrust velocity model.

Figure 17: The inverted velocity for the Overthrust model using (a) the RMSprop, (b) the ADAM, (c) the SGD, (d) the NAG, (e) the L-BFGS and (f) the ML-descent methods. The update for the optimization is performed in the latent space domain and the size of the latent space is 18.

Figure 18: The residuals over iterations for the inversion of the Overthrust model for the various update schemes.

Figure 19: The residual over epochs for training the ML-descent by inverting for the Mar-mousi model.

—

(a)

(b)

Figure 20: The residuals over iterations for the various optimization methods with the latent space size of 18. (a) Marmousi model and (b) Overthrust model. Note that, for curve labelled with ML-descent, the RNN is trained with the quadratic functions only. The curve labeled with ML-descent (transfer) uses the pertained RNN from ML-descent and apply further training using FWI (of the Marmousi model).

Figure 21: The latent space ML-descent optimization of the overthrust model. The inverted model at iterations (a) 10, (b) 15, (c) 20, (d) 25, (e) 30, (f) 40, (g) 50 and (h) 90. Note that the ML-descent algorithm is trained using FWI on the Marmousi model with transfer learning.

Figure 22: The latent space ML-descent optimization of the overthrust model. The inverted model at iterations (a) 10, (b) 15, (c) 20, (d) 25, (e) 30, (f) 40, (g) 50 and (h) 90. Note the ML-descent algorithm here is trained using quadratic functions only.

(a)

(b)

(c)

Figure 23: A typical shot record for the overthrust model. (a) noise free shot record, shot record added with Gaussian random noise for (b) S/N 5 dB and (c) S/N 20 dB.

Figure 24: The inverted velocity for the Overthrust model using (a) the RMSprop, (b) the ADAM, (c) the SGD, (d) the NAG, (e) the L-BFGS and (f) the ML-descent methods. The update for the optimization is performed in the latent space domain and the size of the latent space is 18. The RNN used is that after transfer learning. The S/N of the record for inversion is 5 dB as that shown in Figure 23b.

Figure 25: The inverted Overthrust velocity models using (a) the RMSprop, (b) the ADAM, (c) the SGD, (d) the NAG, e) the L-BFGS and (f) the ML-descent methods. The update for the optimization is performed in the latent space domain with latent space size of 18. We use the RNN trained by the transfer learning. The S/N of the shot record used for the inversion is 20 dB as shown in Figure 23c.

(a)

(b)

Figure 26: The model residuals of the overthrust model over iterations for the various optimization methods in the latent space. The size of the latent space is 18. The curves corresponding to inversion using record of (a) S/N 5 dB, (b) S/N 20 dB. We use the RNN trained by the transfer learning.

Figure 27: (a) P-wave velocity, (b) S-wave velocity, (c) density for the Marmousi-II elastic model and (d) Initial P-wave velocity model for the acoustic FWI. It is a smoothed version of the P-wave velocity model shown in (a).

Figure 28: A typical shot record for the elastic Marmousi-II model.

Figure 29: The inverted velocity for the elastic Marmousi-II model using (a) the RMSprop, (b) the ADAM, (c) the SGD, (d) the NAG, (e) the L-BFGS and (f) the ML-descent methods. The update for the optimization is performed in the latent space domain and the size of the latent space is 18. We use the RNN trained by the transfer learning.

Figure 30: The model residuals of the Marmousi-II model over iterations for the various optimization methods in the latent space: The size of the latent space is 18. The RNN used is that after transfer learning.

Figure A-1: The basic unit of a LSTM cell (Mani, 2019). C_t and h_t are the state variables. x_t is the input. f_t , i_t and o_t corresponds to the forget gate, input gate and output gate, respectively. Refer to appendix A for more information.