A deep learning perspective of the forward and inverse problems in exploration geophysics

Jian Sun*, Zhan Niu, Kris Innanen, Junxiao Li, Daniel Trad

ABSTRACT

Deep learning becomes to be a very powerful and efficient technique in many fields, where the recurrent neural network (RNN) has significant benefits of exhibiting temporal dynamic behavior for time dependency tasks by building a directed graph of a sequence. In this paper, with a self-designed RNN framework, the forward modeling of wave propagation is casted into a forward propagation of RNN, which allows the inversion problem being treated as the training process of RNN. Using this specific network, we numerically analyze the influence and playing role of learning rate (i.e., step-size) for each gradient-based optimization algorithm. Comparisons of gradient-based and non-linear algorithms are also discussed and analyzed. To examine our analysis, the Marmousi model is employed to perform the inversion on the proposed RNN using both gradient-based and non-linear algorithms.

INTRODUCTION

The physical properties of earth can be retrieved by evaluating changes of wave propagation through the source to receivers located either on the surface or in the drilling wells. Hundreds of theories and methods in thousands of literatures have been proposed on approaches of utilizing the gathered seismic information to obtain a better illumination of subsurface structures and to predict more precisely elastic parameters of the interested zone, both in modern seismology and exploration geophysics. One of the most popular methods is full waveform inversion (i.e,. FWI, Lailly and Bednar, 1983; Tarantola, 1984), which uses full wavefield information to reconstruct subsurface parameters by minimizing the misfit between recorded data (the local measurements of seismic wavefield) and the predicted data (a solution of the wave propagation problem using current model). They further showed that, by considering the least-squares minimization, which is not convex, as a local optimization problem.

The gradient, the multidimensional derivatives of the objective function with respect to the model parameters, is usually studied in the framework of Fréchet derivative (McGillivray and Oldenburg, 1990), or Gâteaux derivative. Virieux and Operto (2009) showed that the gradient of the misfit function with respect to the parameters can be built by the crosscorrelation of forward wavefield emitted from the source and the time-reversal wavefield using the data residuals. The nonlinearity of the perturbation wavefield and the perturbed model parameters are usually introduced by the adjoint state techniques (Plessix, 2006; Liu and Tromp, 2006; Yedlin and Van Vorst, 2010) by involving the partial derivatives of the gradient with respect to the model parameters, which is namely called the sensitivity kernel of FWI.

The adjoint state methods are very powerful tool to solve the nonlinear optimizations in the filed of convex problems and proven to be capable of accelerating the convergence of the

objective function for FWI. However, two major problems remain in the implementation of FWI using adjoint state. First, the adjoint state technique requires a prohibitive computational costs, especially when considering the large amounts of subsurface model parameters. To avoid the direct calculation of Jacobian matrix, many iterative approaches are proposed and applied on FWI. For example, the nonlinear conjugate gradient (CG), which the searching direction is a linear combination of the current gradient direction and the previous searching direction (Fletcher and Reeves, 1964). By avoiding explicitly constructing Hessian matrix, the quasi-Newton methods, such as BFGS (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970), provides the approximation of the inverse Hessian using stored models and gradients at previous iterations (Nocedal and Wright, 2006). To reduce the storage requirement and the computation cost of preconditioning, a limited-memory BFGS (L-BFGS) is presented by using limited iterations of recent gradients and models, which provides faster convergence for large-scale bound-constrained optimizations (Byrd et al., 1995; Zhu et al., 1997; Morales and Nocedal, 2011). Another popular approach is truncated Newton (TNC) method, which the searching direction is obtained by solving the Newton system with a matrix-free fashion of conjugate gradient algorithm. Second, the implementation of FWI using adjoint state method is based on the assumption of the convex optimization. This assumption requires a convex optimization problem for which the Hessian matrix is positive semidefinite everywhere and the local minima of the objective function are nearly global minima. To approximately satisfying these conditions, the global optimization problem becomes to be a local optimization, which makes FWI using adjoint state methods heavily relies on the availability of low frequency or high-angle seismic measurements and a good initial model. The detailed discussion is illustrated later in this paper.

Deep learning, also known as deep structured learning or hierarchical learning, is a powerful technique (Dechter, 1986; Schmidhuber, 2015) and becomes to be the most popular method and solutions for variant fields since the successively introduced by Hinton et al. (2006). Deep learning architectures, such as deep neural networks (DNNs), deep belief networks (DBNs) and recurrent neural networks (RNNs), have been applied to fields including computer vision (Cireşan et al., 2012), speech recognition (Hinton et al., 2012a; Graves et al., 2013a,b), natural language processing, audio recognition (Collobert and Weston, 2008), social network filtering (Defferrard et al., 2016), machine translation (Bahdanau et al., 2014), and board game programs (Silver et al., 2016), where they have produced results comparable to and in some cases superior to human experts. The framework of RNN has been shown significant benefits for time-sequence tasks.

To take full advantages of deep learning technique, in this paper, we recast the forward modeling of wave propagation into a RNN framework, which allows us retrieving subsurface model parameters through the training process of neural network. Instead of using Fréchet derivative, we also present the gradient derivation of the objective function with respect to model parameters in a neural network perspective. By illustrating the inversion into a framework of RNN, advance deep learning libraries (such as TensorFlow, PyTorch, Keras, Caffe, and so on) can be directly utilized for geophysical problems. Same to FWI, the training process also employs full wavefield information because the forward propagation of RNN is built by the wave equation. Beyond that, we discuss the efficient and effectiveness of implementing gradient-based optimization algorithms on this inver-

sion problem. The convergence of the objective function and the influence of learning rate (i.e., step-size) for each algorithm is numerically investigated. By theoretically analyzing the learning rate effects on each gradient-based algorithm, we proposed the best appropriate range of learning rate using each algorithm for geophysical velocity inversion problem. Furthermore, the comparison between the best performance of gradient-based methods and CG, L-BFGS, TNC are discussed. Finally, to examine our conclusion, the Marmousi model is employed to perform the inversion process using the proposed RNN framework.

THE FORWARD PROBLEM IN DEEP LEARNING FRAMEWORK

The forward problem

The forward problem, namely, seismic wavefield modeling, usually is being denoted as the partial differential wave equation (Carcione et al., 2002). Several approaches have been proposed and widely applied to discretize the wave equation and model the full seismic wavefields in different types of media, both in time and frequency domains, such as the finite-difference (Smith, 1985; Virieux, 1986) , finite-element (Marfurt, 1984), finite-volume (Eymard et al., 2000; Glinsky-Olivier et al., 2006), pseudo-spectral methods (Faccioli et al., 1997; Etgen and Brandsberg-Dahl, 2009; Li et al., 2018). Assume an acoustic media in 2D with a constant density, the wave equation in time domain is written as,

$$\nabla^2 \mathbf{u}(\mathbf{r},t) = \frac{1}{v^2(\mathbf{r})}\frac{\partial^2 \mathbf{u}(\mathbf{r},t)}{\partial t^2} + \boldsymbol{s}(\mathbf{r},t)\delta(\mathbf{r}-\mathbf{r_s}) \tag{1}$$

where ∇^2 denotes the (spatial) Laplacian operator, and the spatial coordinates is described by $\mathbf{r}$. $\mathbf{u}$ usually represents the pressure or displacement for acoustic medium wave propagation, with the time coordinate t. The source term is denoted by $\mathbf{s}$.

Before casting the forward modeling of wave propagation into a self-designed deep learning framework, first, the wave equation needs to be discretized. To simplify the problem and make it intelligible for readers, the second-order finite difference method, both in time and spatial coordinates, is applied to solve the wave propagation. Therefore, the wavefield at current time step (i.e., $\mathbf{u}_{t+\Delta t}$) is only coherent to the wavefields at two previous time steps (i.e., $\mathbf{u}_t$ and $\mathbf{u}_{t-\Delta t}$). The mathematical formulation is written as,

$$\mathbf{u}(\mathbf{r},t+\Delta t) = v^2(\mathbf{r})\Delta t^2\left[\nabla^2\mathbf{u}(\mathbf{r},t) - \boldsymbol{s}(\mathbf{r},t)\delta(\mathbf{r}-\mathbf{r_s})\right] + 2\mathbf{u}(\mathbf{r},t) - \mathbf{u}(\mathbf{r},t-\Delta t) \tag{2}$$

Equation 2 shows that the forward modeling of wave propagation can be considered as an iterative process which takes the source term $\mathbf{s}(\mathbf{r_s},t)$ and the two previous time steps wavefields as inputs. It inspires us that it is possible to solve the forward modeling problem with a recurrent neural network where each neural network layer does the wavefield modeling at one single time step.

The recurrent neural network

The recurrent neural network (RNN), a class of artificial neural network, builds a directed graph sequence for layers connection. Based on its architecture graph, RNNs are classified into two broad classes of networks: finite impulse using a directed acyclic graph and infinite impulse with a directed cyclic graph, where both of them are capable of exhibiting temporal dynamic behavior for a time sequence (Thulasiraman and Swamy, 2011; Miljanovic, 2012). Unlike feed-forward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to time-dependency signal processing tasks such as unsegmented, connected handwriting recognition or speech recognition (Robinson et al., 2002; Graves et al., 2013a; Sak et al., 2014; Li and Wu, 2015).

Because of the unidirectionality of time flowage in the wave propagation, i.e., the wavefield at current time step is not and will not be infected by the one at next and future time steps, a directed acyclic graph for finite impulse RNN can be considered as a suitable framework for the forward modeling problem. Unlike the conventional RNN architecture, a single cell of our designed RNN for the forward modeling problem represents the finite-difference operator, which takes the sequence at one single time step as the input, outputs the modeled shot record at current time step, and save the memory and the modeled wavefield of this block for the next time step modeling. The directed acyclic graph of RNN architecture is shown in Figure 1. As illustrated, the directed acyclic architecture of RNN can be unrolled, shown in Figure 2. The collection of outputs at every time step sorting into time coordinate, is the local measurements of wavefield, called shot gather in geophysical exploration.

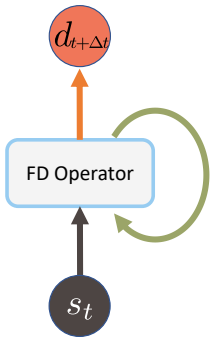

FIG. 1. The directed acyclic graph of RNN for the forward modeling problem.

Based on equation 2, the wavefield at current time step can be obtained by implementing the finite-difference operator on wavefileds at two previous time step and the source term at previous time step, which builds a directed acyclic graph of one single cell of RNN for the forward modeling problem. Note that, similar to the conventional RNNs architecture such as long short-term memory (LSTM) network, a gate factor can be added into the graph to perform the time resampling of the output prediction, i.e., the time step of the RNNs corresponds to the number of iterations in seismic wave forward modeling, where the time resampling is controlled by the gate factor.

The architecture of forward modeling RNN is illustrated in Figure 3. The input $u_{t-\Delta t}$ and u_t represents wavefields at two previous time steps, and the source term at the previous

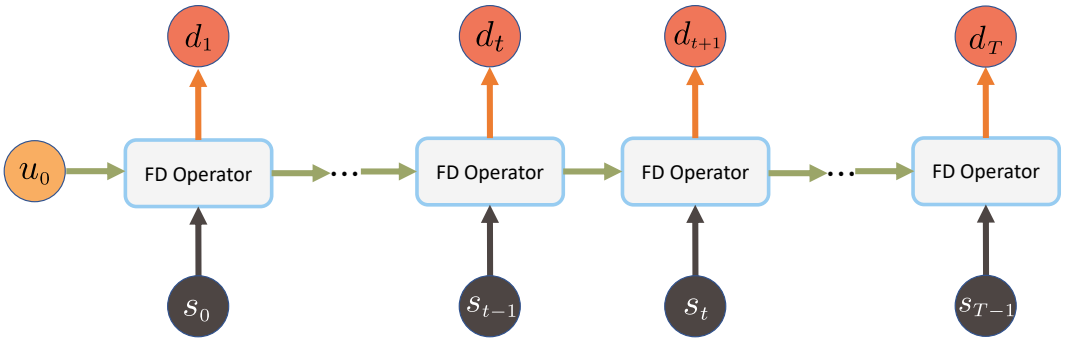

FIG. 2. The unrolled directed acyclic graph of RNNs for the forward problem.

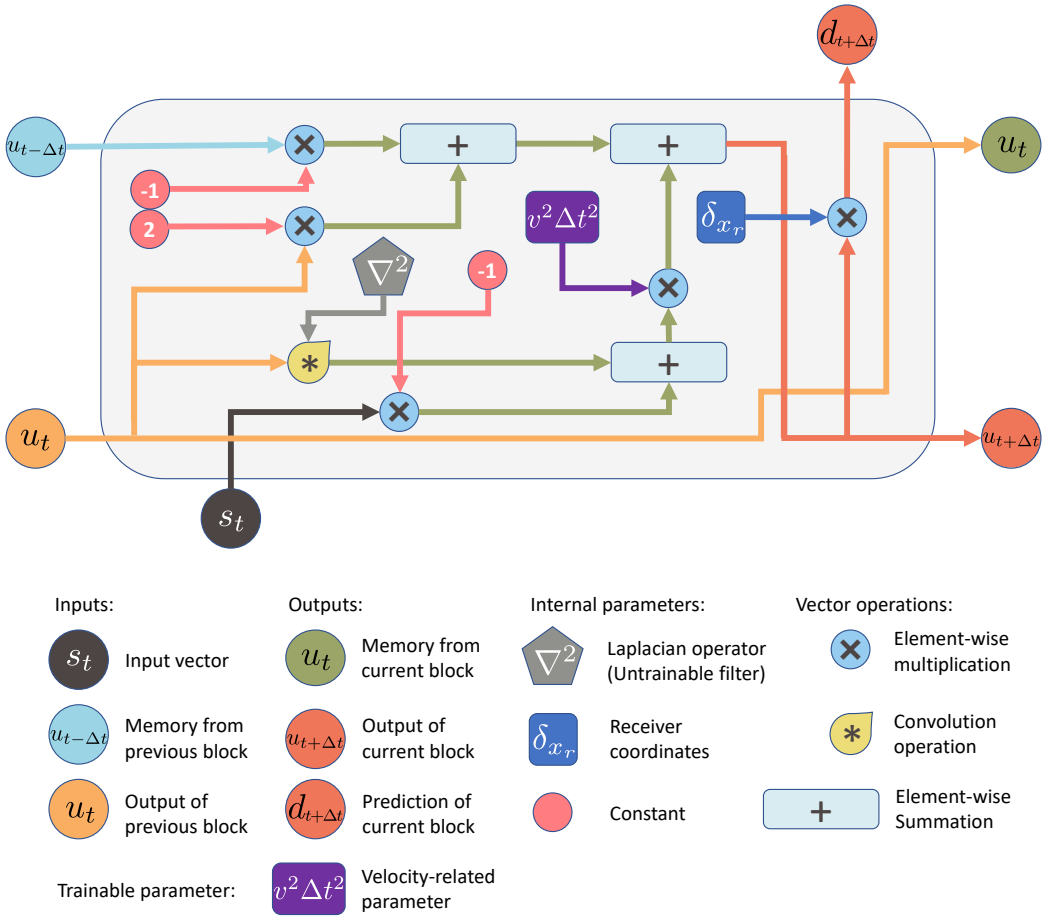

FIG. 3. The single cell's architectures of RNNs for the forward problem.

time step is delineated by s_t. The output $d_{t+\Delta t}$ and $u_{t+\Delta t}$ are the shot record related to receiver locations and the wavefield at current time step, respectively. In Figure 3, the spatial Laplacian operating on the previous wavefield is delineated as a 'convolutional' implementation in neural networks' forward propagation, where the filter (i.e., the spatial Laplacian operator ∇^2, an untrainable filter) is shifting over the entire image (i.e., the previous wavefield u_t). Given the true velocity model, the architecture of RNNs shown in Figure 3 is capable of performing the seismic wave propagation in a perspective of the forward propagation with a RNN framework.

To accommodate the inversion process into a training process of neural networks, the only trainable parameter in the directed acyclic graph of the self-designed RNN is the velocity parameter, which is delineated as the purple unit in Figure 3. The acquired seismic shot records are considered as the labeled data of the training set. To sum up, with the discretized wave equation configured in the self-designed architecture of RNN, the forward modeling problem of seismic wave propagation is casted into the implementation of the forward propagation of RNN. Considering the proper subsurface parameters into the trainable parameters of RNN, the seismic inversion processing turns into the training implementation of RNN, which allows us to measure the underground model parameters in a deep learning perspective.

THE INVERSE PROBLEM IN DEEP LEARNING FRAMEWORK

The inverse problem in geophysical exploration is to obtain the subsurface model parameters based on the approximation of medium type, which is usually being treated as a least-squares local optimization problem by minimizing the square of the misfit between the recorded seismic records and the modeled seismic data, i.e., $\Delta\mathbf{d} = \mathbf{d_{obs}} - \mathbf{d_{pred}}$. In time domain, the implicitly of summation is performed over the number of source-channel pairs (where the channel is related to the component of sensor), the number of receivers for one shot, and the number of time samples in recorded seismograms. To further analyze the gradient calculation for model parameters updates in a deep learning framework, instead of the vector operations, we need to expand the recorded and modeled data into a time-related sequence vector. The mathematical formulation of the least-square norm of the misfit function or the objective function is written as,

$$J(v) = \frac{1}{2n_s} \sum_{\mathbf{r_s}} \sum_{\mathbf{r_g}} \sum_{\mathrm{t}} (\mathbf{d}_t - \tilde{\mathbf{d}}_t)^2 \tag{3}$$

where, $\mathbf{r_s}$ represent the source location, $\mathbf{r_g}$ is the receiver location related, t delineates the propagating time step, and n_s denotes the number of sources. $\mathbf{d}_t$ is a vector of source-related recorded seismic record at a fixed time step, $\tilde{\mathbf{d}}_t$ is a vector of source-related modeled seismic record. Both of them are coherent to receiver locations, which can also be written as the multiplication of a delta function of receiver-location and the corresponding wavefield at the same time step, i.e., $\tilde{\mathbf{d}}_t = \delta_{\mathbf{r_g}} \tilde{\mathbf{u}}_t$.

In modern geophysical exploration and application, the general way to solve the least-squares optimization problem illustrated in equation 3 is the gradient-based method. Suppose that at the n-th iteration, the model at next iteration $v_{n+1}(\mathbf{r})$ is updated with the current model $v_n(\mathbf{r})$ through a perturbation model $\delta v_n(\mathbf{r})$

$$v_{n+1}(\mathbf{r}) = v_n(\mathbf{r}) + \alpha \delta v_n(\mathbf{r}) \tag{4}$$

where α represents step-size at the current iteration, which is usually being called as learning rate in deep learning perspective. In the following contents, to be consistent, the learning rate is used.

The first-order optimization

The calculation of the perturbation model $\delta v_n(\mathbf{r})$ varies depending on the approximate order of the optimization algorithms. Using the first-order approximate optimization algorithms, the model update is usually achieved by the negative gradient, known as steepest descent or gradient descent (GD) algorithm. The mathematical formulation of model perturbation using gradient descent method is written as

$$\delta v_n(\mathbf{r}) = -g_n(\mathbf{r}) \tag{5}$$

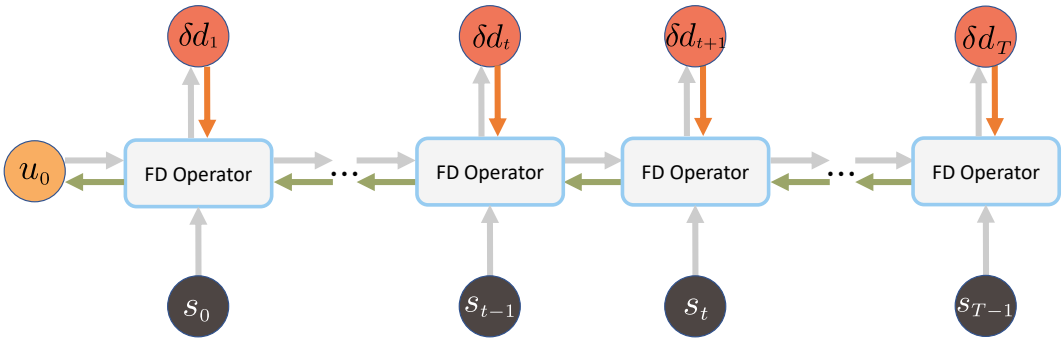

FIG. 4. The unrolled directed acyclic graph of RNNs for the inverse problem.

In perspective of RNN's framework, the gradient calculation of the objective function over the trainable parameter is achieved by the chain rule performing in sense of reversal time, shown in Figure 4. The mathematical formulation of the gradient is written as (detailed derivation can be found in Appendix A),

$$\begin{aligned} g &= \frac{\partial J}{\partial v} \\ &= BP(-\frac{1}{n_s \boldsymbol{v}^2 \Delta t^2} \sum_{\mathbf{r_s}} \sum_{\mathbf{r_g}} \boldsymbol{\delta} \mathbf{d_t}) \frac{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}}{\partial \boldsymbol{v}} \\ &\approx BP(-\frac{1}{n_s} \sum_{\mathbf{r_s}} \sum_{\mathbf{r_g}} \boldsymbol{\delta} \mathbf{d_t}) \frac{2}{\boldsymbol{v}^3} \frac{\partial^2 \tilde{\boldsymbol{u}}_{\boldsymbol{t}}}{\partial t^2} \end{aligned} \tag{6}$$

where, $BP(s)$ indicates the back-propagation of source s in reversal time ($T \rightarrow 0$). $\boldsymbol{\delta} \mathbf{d_t}$ represents the residuals between observed and predicted data. The initial states for the back-propagation are,

$$\left[\frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}}\right]_{t=T} = -\frac{1}{n_s} \sum_{\mathbf{r_s}} \sum_{\mathbf{r_g}} (\mathbf{d_T} - \boldsymbol{\delta}_{\mathbf{r_g}} \tilde{\mathbf{u}}_{\mathbf{T}}) \tag{7a}$$

$$\left[\frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}}\right]_{t=T-1} = -\frac{1}{n_s} \sum_{\mathbf{r_s}} \sum_{\mathbf{r_g}} [(\boldsymbol{v}^2 \Delta t^2 \nabla^2 + 2) \boldsymbol{\delta} \mathbf{d_T} + \boldsymbol{\delta} \mathbf{d_{T-1}}] \tag{7b}$$

Equation (6) indicates that the perturbation model obtained by the gradient is performed by the crosscorrelation between the second-order partial derivative of the forward wavefield over time and the back-propagation wavefield using the residual as the source, which is equivalent to the gradient of time-domain full-waveform inversion shown in paper by Yang et al. (2015). In other words, FWI is also a specified machine learning process with a self-designed RNN framework, where the unknown subsurface parameters are treated as the trainable weights of neural network.

The first-order gradient-based optimization algorithm has been proved as an efficient and effective method that was of core practical importance in fields of machine learning science and engineering. However, gradient decent may lead to slower-learning, or even divergence optimization results when the noisy stochastic objectives is considered which frequently happened in practical datasets. To solve that, the efficient stochastic optimization techniques are required. Gradient with momentum (Momentum) was first proposed by Qian (1999) to optimize the direction of convergence and accelerate the speed of gradient descent method. Duchi et al. (2011) proposed an adaptive sub-gradient (Adagrad) method by scaling gradient with its squared norm, which works well for sparse gradients of stochastic optimization. Zeiler (2012) presented an adaptive robust learning rate (Adadelta) method based on gradient decent, which does not require the manual tunning and section of hyperparameters. Around the same time, Hinton et al. (2012b) also proposed a similar way to scale the gradient by running the magnitude of recent gradients normalization, which is called the root-mean-square gradient (RMSprop) algorithm. Inspired by Momentum and RMSprop algorithms, Kingma and Ba (2014) combined these two methods as the adaptive moment (Adam) estimation, where the magnitude of parameter updates are invariant to rescaling the gradient, it does not require a stationary objective and works well with sparse gradients, and it naturally performs a form of step-size annealing. To help reader further understand these algorithms, the pseudo-codes of Momentum , Adagrad , RMSprop and Adam algorithms are detailed reviewed in Appendix B.

The second-order optimization

As delineated in introduction, to accelerate the convergence of minimization of objective function, FWI is usually implemented with the adjoint-state method, which are second-order approximation based. The updates for model parameters equal to the multiplication of negative Hessian matrix and gradient, which is written as,

$$\delta v_n(\mathbf{r}) = -\sum_{\mathbf{r'}} H_n^{-1}(\mathbf{r}, \mathbf{r'}) g_n(\mathbf{r'}) \tag{8}$$

where $g_n(\mathbf{r})$ and $H_n^{-1}(\mathbf{r}, \mathbf{r'})$ are the gradient and the Hessian matrix at the n-th iteration, respectively. Their observation is performed as the first- and second-order derivatives of the objective function $J(v_n)$ with respect to the model v_n, respectively,

$$
\begin{aligned}
g_n(\mathbf{r}) &= \frac{\partial J(v_n)}{\partial v(\mathbf{r})} \\
H_n^{-1}(\mathbf{r}, \mathbf{r'}) &= \frac{\partial^2 J(v_n)}{\partial v(\mathbf{r}) \partial v(\mathbf{r'})}
\end{aligned}
\tag{9}
$$

The second-order optimization algorithms in equation 8 become to be the most powerful and popular way to solve the specialized optimization in the field of convex problems for which the Hessian matrix is positive semidefinite everywhere. The objective functions of such optimization problems are well-behaved, lacking of saddle points, and all their local minima are necessarily global minima (Goodfellow et al., 2016). However, most problems in deep learning are difficult to express in terms of convex optimization, as well as FWI. Some challenges arise even when optimizing convex deep learning problems which the most prominent is ill-conditioning of the Hessian matrix. For example, the the Taylor series expansion of function $f(\mathbf{x})$ around the current point $\mathbf{x^0}$,

$$
f(\mathbf{x}) = f(\mathbf{x^0}) + (\mathbf{x} - \mathbf{x^0})^* \boldsymbol{g} + \frac{1}{2}(\mathbf{x} - \mathbf{x^0})^* \boldsymbol{H} (\mathbf{x} - \mathbf{x^0}) + \mathcal{O}(\mathbf{x}) \tag{10}
$$

where $\boldsymbol{g}$ and $\boldsymbol{H}$ are the gradient and Hessian matrix at $\mathbf{x^0}$, * represents conjugate transpose. With the gradient descent method and a learning rate α, the prediction for $\mathbf{x}$ position is given by $\mathbf{x^0} - \alpha \boldsymbol{g}$. Neglect the high order approximation, equation 10 can be rewritten as

$$
f(\mathbf{x^0} - \alpha \boldsymbol{g}) \approx f(\mathbf{x^0}) - \alpha \boldsymbol{g}^* \boldsymbol{g} + \frac{1}{2} \alpha^2 \boldsymbol{g}^* \boldsymbol{H} \boldsymbol{g} \tag{11}
$$

Equation 11 indicates that, with the second-order optimization algorithm, the cost function correction at position $\mathbf{x^0}$ is evaluated by $\frac{1}{2}\alpha^2 \boldsymbol{g}^* \boldsymbol{H} \boldsymbol{g} - \alpha \boldsymbol{g}^* \boldsymbol{g}$. In many cases, the gradient norm $\boldsymbol{g}^* \boldsymbol{g}$ does not decrease significantly through learning, but the $\boldsymbol{g}^* \boldsymbol{H} \boldsymbol{g}$ term grows by more than an order of magnitude (Goodfellow et al., 2016). This leads to a very slow learning process even in presence of a large gradient because a much smaller learning rate must be taken to compensate for even stronger curvature. Therefore, for a non-convex optimization problem or with noisy labeled data or a bad initial model, the second-order approximation based optimization algorithms can not guarantee a faster and smoother convergence than the first-order gradient-based algorithms. For these reasons, the most widely utilized optimization algorithm in deep learning problems is first-order derivative based method, and the searching of perturbation model is in the negative direction of gradient.

However, it has been proved that the adjoint state method is able to accelerate the convergence process when it starts with a well-behaved initial model in FWI problem. Therefore, in the following section of this paper, we try to compare the performance of the first-order (includes GD, Adagrad, RMSprop, and Adam) and second-order (includes non-linear CG, L-BFGS, and TNC) based algorithms on this geophysical optimization problem. The numerical analysis of the learning rate effects, the speed of convergence using varied optimization algorithms are also discussed using a one-dimension (1D) depth-varying model.

NUMERICAL ANALYSIS OF HYPERPARAMETER SELECTION

A key problem in deep learning training process is the hyperparameter tuning, which is still one of the toughest obstacles. In general deep learning cases, the learning rate is usually determined by empirical analysis or some trials, which is in range of $(0, 1]$. As we introduced, the proposed RNN in this paper is not a typical architecture presented in published literatures. Does the $[0, 1)$ scope of learning rates for different algorithms works for the geophysical RNN framework presented in previous section? To investigate the effect of the hyperparameter in deep learning framework for geophysical inversion and how to select a suitable learning rate for difference optimization algorithms, in this section, some trials experiments are shoot based on a depth-varying velocity profile. The influences of different learning rates on variant optimization algorithms, and the comparison between first-order and second-order optimization algorithm are discussed.

The depth-varying model

In Figure 5a, a 4-layer velocity profile, from top to bottom: $[2000, 3000, 4000, 5000]$m/s is illustrated as back line. Using the second-order finite difference method, the seismic record is generated for later inversion, shown in Figure 5b. All inversion using different optimization algorithm starting with the same initial velocity model plotted in red in Figure 5a.

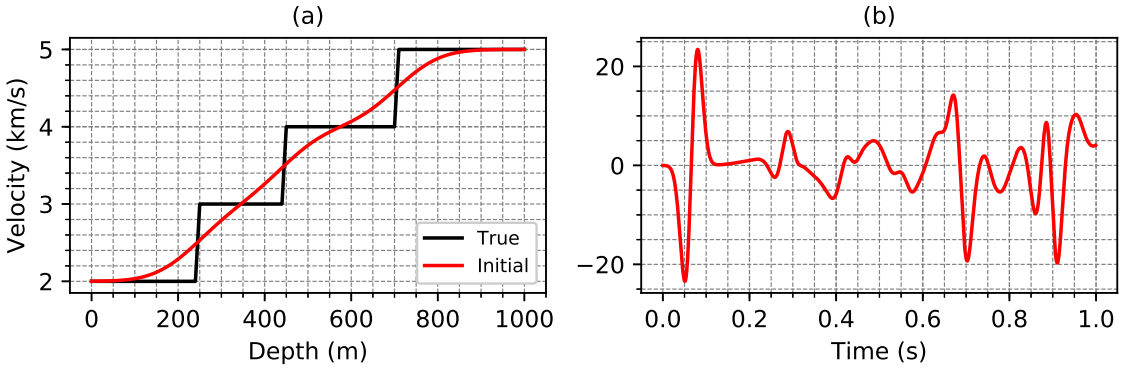

FIG. 5. A depth-varying model. (a) Back: the 1D true velocity model. Red: the initial velocity model for inversion. (b) Synthetic seismic records generated using second-order finite difference method based on the depth-varying model which is delineated as black in left panel.

Inversion with gradient-based algorithms

To investigate the efficiency of several most popular gradient-based optimization algorithms in deep learning problems, we implement the training of proposed RNN using GD, Momentum, Adagrad, RMSprop, and Adam. First, some trial experiments are shoot for hyperparameter tunning (i.e., learning rate) to find the best performance for each gradient-based algorithm. The appropriate ranges of learning rate for each gradient-based algorithms on this geophysical velocity inversion are also studied and theoretically analyzed. After that, the comparisons of the best performances using GD, Momentum, Adagrad, RMSprop, and Adam on RNN inversion are detailed discussed.

Gradient descent

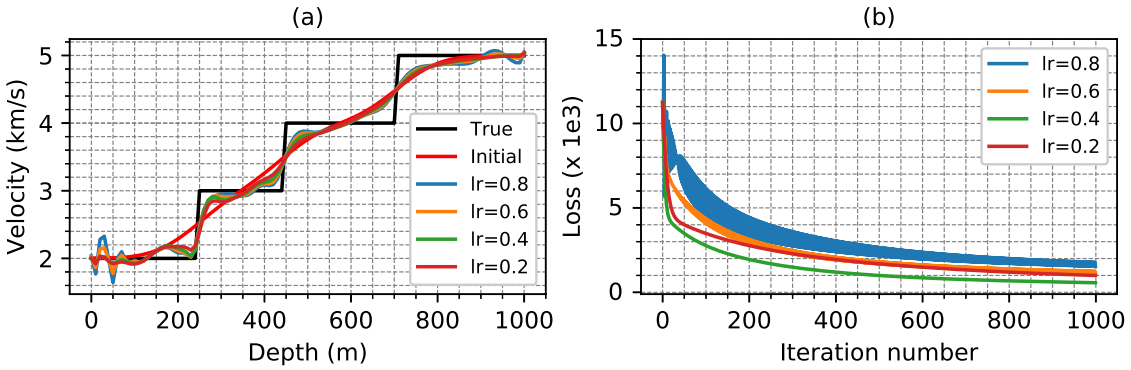

FIG. 6. The inversion with gradient descent algorithm. (a) The predicted results of gradient descent with different learning rates, where the true model is indicated as black and the initial model is in red. (b) The values of objective function with respect to iteration numbers, using the corresponded learning rates in panel (a).

In Figure 6, the inversion results using GD with variant learning rates are delineated, where the final inverted model and the corresponding loss values with respect to varying iteration numbers are plotted in panel (a) and (b), respectively. The inversion is implemented using GD with four trial learning rates $[0.2, 0.4, 0.6, 0.8]$ and the maximum iteration number is 1000. Figure 6b indicates that the oscillation occurred in the inversion process using GD with learning rate equaling to 0.8 and 0.6. Instead of the slow learning process with 0.2, a value of 0.4 for learning rate gives the slight faster and smoother convergence of the objective function using GD. However, as shown in Figure 6a, only the first interface is visible after 1000 iterations with these four proposed learning rates. It means, for casted geophysical inversion problem in RNN framework, GD requires large amounts of iterations to converge the objective function and also demands some trial experiments to determine the best learning rate. Remember that, the appropriate scope of learning rate for GD is $(0, 1]$.

Gradient with momentum

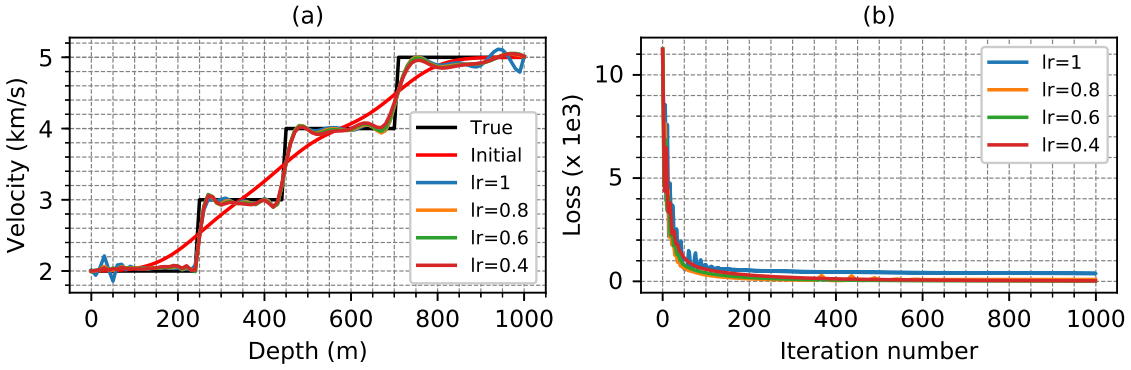

FIG. 7. The inversion with momentum algorithm. (a) The predicted results of momentum using different learning rates, where the true model is indicated as black and the initial model is in red. (b) The values of objective function with respect to iteration numbers, using the corresponded learning rates in panel (a).

The momentum is also a gradient-based method which is developed to accelerate the stochastic gradient descent in the relevant direction and dampens oscillations. Unlike the gradient descent where the model is updated with the negative gradient directly, momentum strengthen the effects of preciously obtained gradients by accumulating a scaled version of them which is controlled by the hyperparameter β. As the algorithm shown in Appendix B, the momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. The adapted learning rate is similar to the gradient descent, which is in range of $(0, 1]$. As a result, we gain faster convergence and reduced oscillation.

In Figure 7, we implemented the RNN inversion using Momentum algorithm with a list of learning rates: $[0.1, 0.6, 0.8, 1]$. Comparing to results using GD shown in Figure 6, the inversion with Momentum converges much faster and provides much better results agreeing with the true velocity profile. Besides that, the oscillations are also reduced for the same learning rates applied on GD. Figure 7b shows that small oscillations occurred with Momentum algorithm at the first few iterations, however, it has no significant influence on the convergence. It's worth to note that Momentum with learning rate equaling to $[0.4, 0.6, 0.8]$ are converged to a similar value with approximately same speed, except learning rate equaling to 1.0. This indicates that the speed of convergence and the final predicted results are not determined by the value of the learning rate, which is due to the accumulated gradients in Momentum.

Adaptive gradient

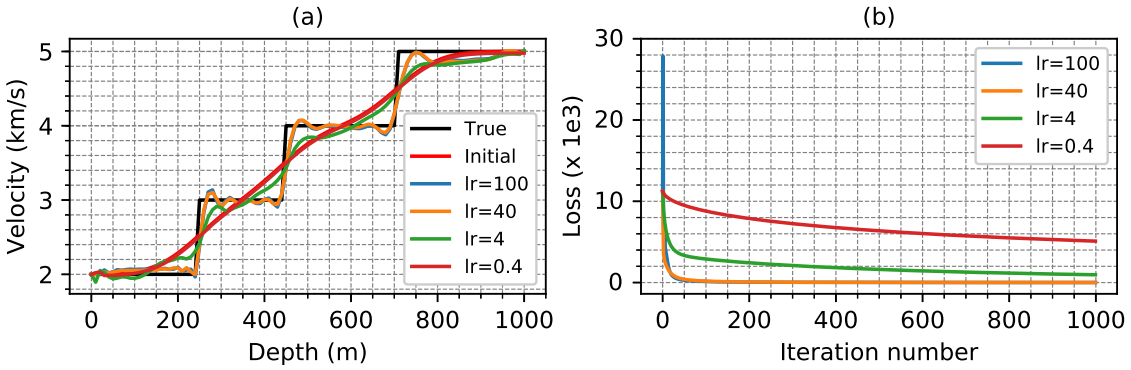

FIG. 8. The inversion with Adagrad algorithm. (a) The predicted results of Adagrad using different learning rates, where the true model is indicated as black and the initial model is in red. (b) The values of objective function with respect to iteration numbers, using the corresponded learning rates in panel (a).

One of the most popular methods in deep learning cases is the adaptive gradient method which provides an adjustive version of gradient descent using previously accumulated squared gradients (see algorithm 2 in Appendix B). Here, according to the analysis by Duchi et al. (2011), the value of hyperparameter β is determined to be a fixed value 0.9 for all following experiments. A collection of learning rate $[0.4, 4, 40, 100]$ are selected for trial experiments using Adagrad method. All inversion results and related values of objective function are plotted in Figure 8a and 8b, respectively.

Figure 8 indicates that, with a small learning rate (0.4), the Adagrad algorithm is able to smoothly decrease the objective function. However, the speed of learning seems to be relatively slow and it would requires much more than 1000 iterations to converge the inversion process. This is verified in Figure 8a where the final predicted profile using learning rate 0.4 after 1000 iterations is nearly the initial velocity model. By increasing the learning rate ($0.4 \rightarrow 40$), the speed of inversion process using Adagrad method is much faster while the loss value keeps smoothly decrease. With learning rate equaling to 100, the Adagrad is still able to converge quickly at around same value of objective function, but with oscillations occurred at the fist few iterations. One question is, why does Adagrad allow a learning rate much larger than 1, which is unusual in general deep learning cases.

In most of deep learning cases, the activation function is always applied in forward propagation of predicts and in calculation of residuals between labeled and predicted data. The operation of activation function will cause the training parameters are usually in range of $[0, 1]$. With the normalized random initialization, the perturbations obtained from gradient is in range of $[-1, 1]$. This means, in general deep learning cases, there is no magnitude difference of the residuals and the accumulated squared-norm of gradients. However, this does not stands for RNN geophysical inversion problems. The magnitude difference between initial velocity and the true model may be $0 - to - 100$ in velocity inversion considering the required maximum iteration for convergence less than 100. For example, at the k iterations, the model parameters update is $\alpha \cdot \boldsymbol{\delta v}_k / \sqrt{\boldsymbol{G}_k + \epsilon}$, where $\boldsymbol{G}_k$ is the accumulated squared norm of gradients. Because the gradient $\boldsymbol{\delta v}_k$ is in the same magnitude order of $\sqrt{\boldsymbol{G}_k + \epsilon}$, the absolute norm of model update being in range of $[0, \alpha * const]$ where $const \in [0, 10)$. Therefore, by limiting the model update at each iteration, a small learning rate for Adagrad may suffers from slowing learning rate.

To accelerate the speed of convergence and still take advantages of the searching direction provided by Adagrad, we need a learning rate which can balance the magnitude of $\sqrt{\boldsymbol{G}_k + \epsilon}$. An appropriate learning rate using Adagrad satisfies $0 < \alpha / \sqrt{\boldsymbol{G}_k + \epsilon} \approx \alpha / \boldsymbol{\delta v}_1 \leq 1$. This leads to the best appropriate range of learning rate for Adagrad is $\alpha \in (0, 100]$. It is worth to note that, the model updates at the first iteration is a constant of learning rate because of the initial state for diagonal matrix $\boldsymbol{G}_0 = 0$, which explains a very large learning rate may cause oscillation at the first few iterations. The interesting point is the model update automatically shrinks because of the accumulated squared scale of gradients. It means a very large learning rate does not affect the convergence of the objective function, and may also accelerate the speed of the convergence even with small oscillations happened in first few iterations. This conclusion is proven in Figure 8. With the learning rate 100, the loss values oscillated after the first few iterations and converged very quickly (Figure 8b), while the inversed result is well-behaved and agrees with the true velocity profile, shown in Figure 8a. To successfully applying the Adagrad method on geophysical inversion, we recommend the range of $[10, 100]$ for the learning rate, which could accelerate the convergence and achieve the well-matched inversion results. To emphasize, within the proposed range for learning rate, a small value provides slower but smoother convergence process, while a relative large learning rate offers much faster convergence but oscillations occurred at first few iterations. Both of these learning rate is able to converge the objective function because of the automatically tuning of the step-length of model updates which is brought by the accumulated squared-norm of gradients.

RMSprop & Adadelta

Both RMSprop and Adaptive delta (i.e., Adadelta, an extension of Adagrad) were developed independently to resolve the radically diminishing learning rates caused by Adagrad method. These two methods are very similar and identical to each other under some circumstances. To avoid excessive repetitions, the trial experiments are only implemented using RMSprop. Similar to Adagrad algorithm, RMSprop is also in a way to scale the gradient. Instead of directly using accumulated squared-norm of gradients along iterations, a hyper-factor is considered to weaken the influence of the accumulated squared gradients. Hinton et al. (2012b) indicated that the hyperparameter is usually being a fixed value $\beta = 0.9$. Therefore, at $k - th$ iteration, the model updates using RMSprop is $\alpha \cdot \boldsymbol{\delta v}_k / (\sqrt{\tilde{\boldsymbol{r}}_k} + \epsilon)$. The appropriate scope for the learning rate using RMSprop is $0 < \alpha / (\sqrt{\tilde{\boldsymbol{r}}_k} + \epsilon) \approx \alpha / (\sqrt{1 - \beta} \boldsymbol{\delta v}_1) \leq 1$, which leads to the best approximately range for learning rate: $\alpha \in (0, 10]$. Again, the RMSprop inherits benefits from Adagrad, i.e., the step-length of model updates is auto-adjustive. Therefore, to speed up of the convergence of the objective function for RNN velocity inversion, we may conclude that the best learning rate range using RMSprop is $[1, 10]$.

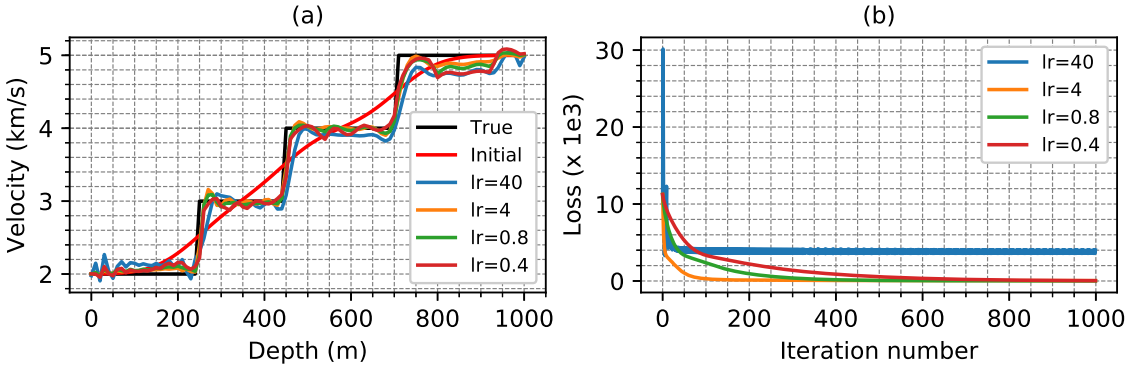

FIG. 9. The inversion with RMSprop algorithm. (a) The predicted results of RMSprop using different learning rates, where the true model is indicated as black and the initial model is in red. (b) The values of objective function with respect to iteration numbers, using the corresponded learning rates in panel (a).

In Figure 9, we plotted the inversion results and the loss values with respect to varying iteration numbers, using the learning rate $[0.4, 0.8, 4, 40]$ in panel (a) and (b), respectively. As expected, in Figure 9b, a very larger learning rate ($lr > 10$, i.e., 40) causes oscillations and divergence problems using RMSprop algorithm, and a very small learning rate ($lr < 1$, i.e., 0.4 and 0.8) requires amounts of iterations to converge the objective function. With a learning rate in range of $[1, 10]$, RMSprop is capable of providing a much stable and faster convergence processing and a well-agreed inversion result, shown in Figure 9a.

Adaptive moment

To take advantages of Momentum and RMSprop algorithms, Adam is proposed that computes adaptive scaling for each model parameter. As illustrated in algorithm 4 of Appendix B, the model updates without learning rate is $\tilde{\boldsymbol{m}}_k / (\sqrt{\tilde{\boldsymbol{r}}_k} + \epsilon)$, which ensures that the update is always normalized. This works well in general deep learning problems, be-

cause the activation function limits the gradient and the model update in the same range of $[0, 1]$. However, as pointed out, the model updates for geophysical inversion is much larger and usually in magnitude of $[1, 100]$. In other words, with the learning rate $(0, 1]$, Adam requires amounts of iterations to converge the objective function. We found that this issue can be easily solved by tuning a relative larger learning rate. And the maximum value for model updates is restricted by the learning rate α, for example, the limit of the model updates at one single iteration using Adam is $-\alpha < modelUpdates < \alpha$. An appropriate learning rate should satisfy: $0 < \alpha \cdot (1-\beta_1)/(\sqrt{1-\beta_2} \cdot \boldsymbol{\delta v}_1) \leq 1$, i.e., $\alpha \in (0, 100]$. Again, to maximum accelerate the convergence process, we recommend the appropriate range of learning rate using Adam: $[10, 100]$. Inheriting from RMSprop, Adam algorithm also has the ability to shrink the 'step-length' of model updates during the iteration process, which means a relatively large learning rate can also be tolerated with oscillation occurred at first few iterations.

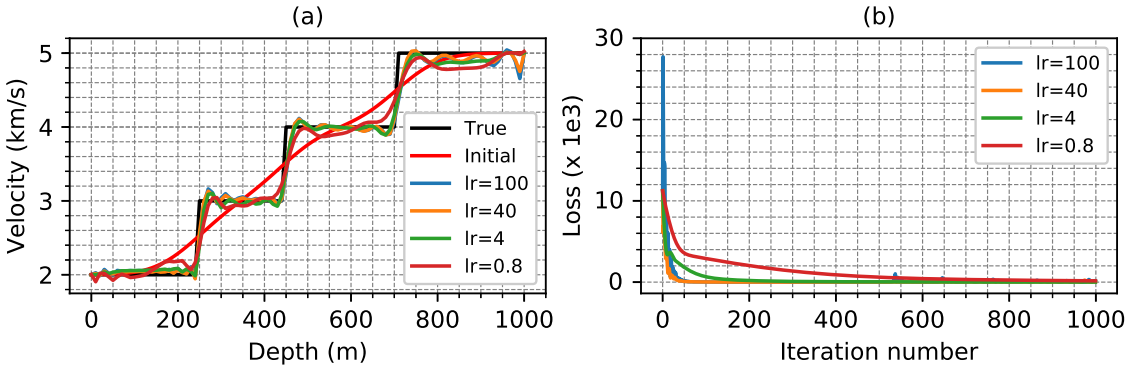

FIG. 10. The inversion with Adam algorithm. (a) The predicted results of Adam using different learning rates, where the true model is indicated as black and the initial model is in red. (b) The values of objective function with respect to iteration numbers, using the corresponded learning rates in panel (a).

To examine our inferences of Adam, in Figure 10, the inversion results and the loss values using a list of learning rates $[0.8, 4, 40, 100]$ are plotted in panel (a) and (b), respectively. Figure 10a shows all inversion results with listed learning rates are well matched to the true velocity profile within 1000 iterations. In Figure 10b, the value of objective function with respect to iteration numbers convince that the learning rate only affects the convergence speed of the objective function. Same to RMSprop method, a very large learning rate causes oscillations at first few iterations, but Adam is able to quickly and smoothly converge after that because of its capacity of automatically shrinking the 'step-length' of model updates.

Comparisons of GD, Momentum, Adagrad, RMSprop, and Adam

To intuitively observe differences of gradient-based algorithms discussed, their best performances with appropriate learning rates are plotted in Figure 11. Recall that, the appropriate range of learning rate for each algorithm is $(0, 1]$ for GD and Momentum, $[1, 10]$ for RMSprop, and $[10, 100]$ for Adagrad and Adam. For this 1D inversion case, based on trial experiments we have, the most appropriate learning rate for each algorithm is 0.4 (GD), 0.8 (Momentum), 40 (Adagrad), 4 (RMSprop), 40 (Adam).

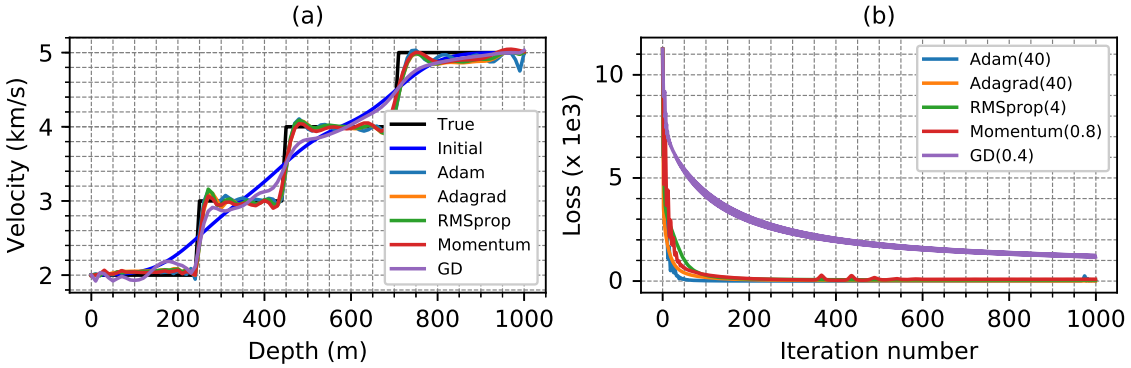

FIG. 11. Comparison of best performance using variant gradient-based algorithms. (a) The best inversion results using variant gradient-based algorithms with their appropriate learning rate, such as GD: 0.4, Momentum: 0.8, Adagrad: 40, RMSprop: 4, Adam:40. (b) The values of objective function with respect to iteration numbers using illustrated gradient-based methods.

In Figure 11a, the results show that all gradient-based algorithms are capable of achieving acceptable inversion results with less than 1000 iterations except for GD. Figure 11b shows that GD has the slowest convergence speed comparing to the rest of gradient-based algorithms. With appropriate learning rate, Momentum, Adagrad, and RMSprop are able to converge within 200 iterations. Adam shows the fastest speed of convergence, which is less than 50 iterations. Note that, the value of the learning rate in proposed appropriate range using Adam only mainly affect the convergence speed.

Inversion with non-linear optimization algorithms

To investigate the performance using the second-order optimization algorithms, we also implement the 1D RNN inversion using non-linear CG, L-BFGS, and TNC methods. The inversion result and the value of objective function using non-linear CG are plotted in Figure 12a and Figure 12b, respectively. It is clear that, in Figure 12a, CG provides a very well-agreed inversion results comparing to the true velocity profile. In Figure 12b, the iteration number delineates the number of decreasing the objective function, which does not equal to the external iteration of non-linear CG. However, the computational cost time of one-single iteration (not external iteration of CG) we presented is approximately equivalent to the computational cost time of each iteration using gradient based algorithms. This fact also stands for L-BFGS algorithm. Figure 12b shows that, comparing to GD, non-linear CG is very efficient and much faster at objective function convergence on the 1D RNN inversion case.

Figure13 shows the final prediction and value of objective function using L-BFGS method. In Figure 13a, the completed inversion using L-BFGS algorithm is very competitive to the one with non-linear CG. Beyond that, L-BFGS takes approximately half of time which non-linear CG takes to converge the objective function, shown in Figure 13b. One can say that both of these two method are efficient to converge the objective function and generate well-matched inversion results. However, the inversion process using TNC shows strong oscillations plotted in Figure 14b. And the final predicted result has significant bias at interfaces.

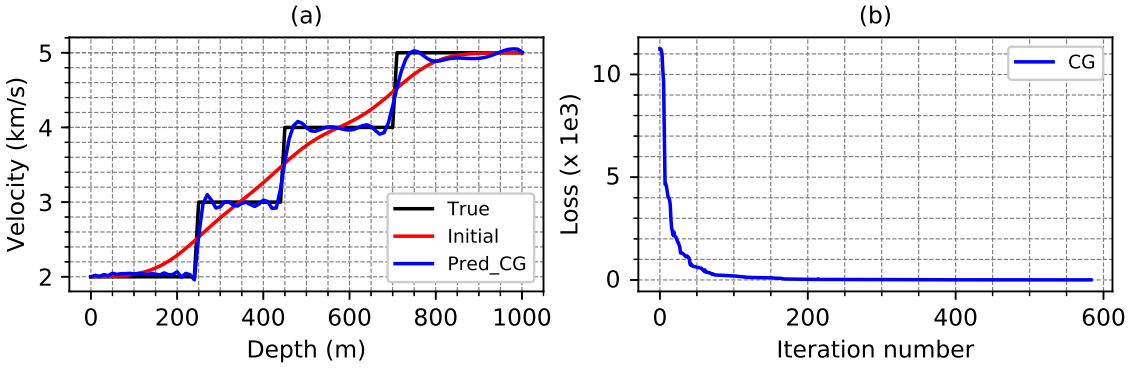

FIG. 12. The inversion with non-linear CG algorithm. (a) The predicted results of non-linear CG algorithm indicated as blue, the true model is indicated as black and the initial model is in red. (b) The values of objective function with respect to iteration numbers.

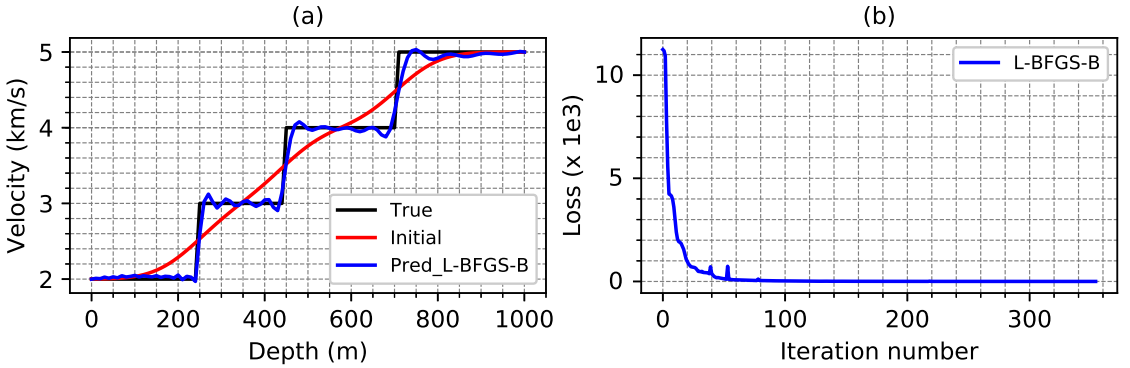

FIG. 13. The inversion with L-BFGS algorithm. (a) The predicted results of L-BFGS algorithm indicated as blue, the true model is indicated as black and the initial model is in red. (b) The values of objective function with respect to iteration numbers.

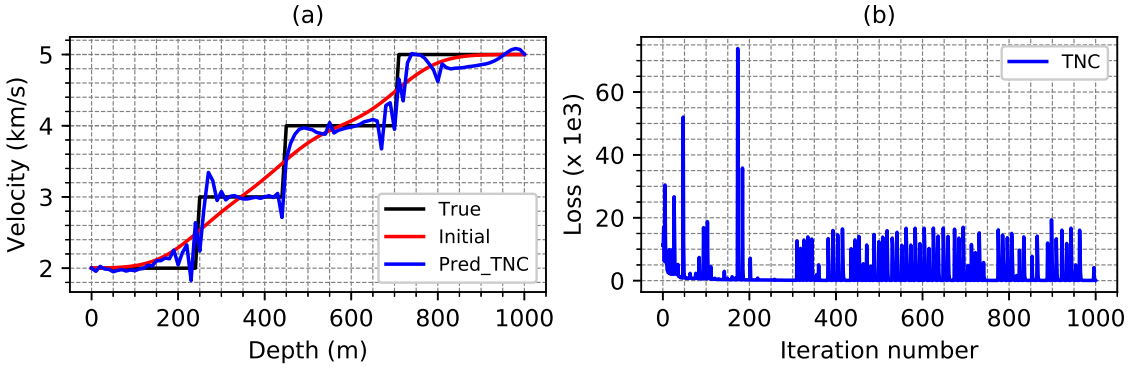

FIG. 14. The inversion with TNC algorithm. (a) The predicted results of TNC algorithm indicated as blue, the true model is indicated as black and the initial model is in red. (b) The values of objective function with respect to iteration numbers.

To investigate and further analyze the differential performance provided by gradient-based and second-order approximation algorithm, we select the best performance of GD and Adam with their appropriate learning rate. The comparisons between GD, Adam, CG, and L-BFGS are plotted in Figure 15. As denoted in Figure 15a, Adam is able to

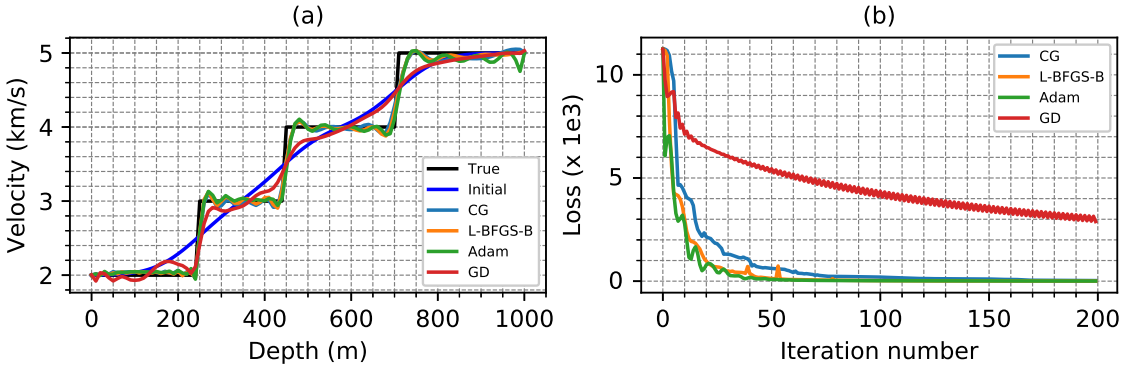

FIG. 15. Comparisons of GD, Adam, CG, and L-BFGS algorithms. (a) The best inversion results using GD, Adam, CG, and L-BFGS. (b) The values of objective function with respect to iteration numbers.

obtain well-agreed inversion results as non-linear algorithms do, while GD requires more iterations to obtain a better prediction. Moreover, on this 1D RNN case, the convergence speed of objective function using Adam is similar to L-BFGS method and faster than non-linear CG algorithm.

THE SYNTHETIC EXAMPLE OF MARMOUSI

To fully examine the capacity of casted RNN framework for geophysical inversion, in this section, the 2D synthetic Marmousi model is employed to create synthetic short records as observed data, and a smoothed model is considered as the initialization for all training parameters in model coordinates. The true velocity of Marmousi model is shown in Figure 16a. Note the labels of depth and lateral positions are not consistent with the original size of Marmousi because we shrink the grid interval due to the consideration of computational memory limitation on the laptop.

As discussed, the forward modeling can also be achieved with this self-designed RNN framework by initializing the trainable parameters with the true velocity model. Using the true Marmousi model shown in Figure 16a, we generate 12 shot gathers while the source location moves from left to right with the interval 25m at depth of 40m. The first 10 of 12 shot records are displayed in Figure 17. For comparison, both Adam, CG, and L-BFGS are employed to implement the RNN inversion of Marmousi model. The same initial velocity profile is utilized for all algorithms and plotted in Figure 16b.

Based on the numerical analysis in previous section, for Adam algorithm implementation, the learning rate has no significant effects on final inversion results and only matters the speed of convergence. And considering oscillations during first few iterations, the appropriate range of the learning rate for Adam is $[10, 100]$. Here, we choose 40 for learning rate with Adam on inversion of 2D Marmousi model. The inversion results using Adam algorithm at $[25, 50, 100]th$ iterations are delineated in Figure 18. It's fair to say that visible structures and robust layer information of Marmousi model are predicted using Adam after 25 iterations. With 50 iterations, Adam has the capacity of achieving most information of structures and layers at shallow zones. After 100 iterations, the information of layers and structures at deep zones are well predicted and consistent with the true Marmousi model

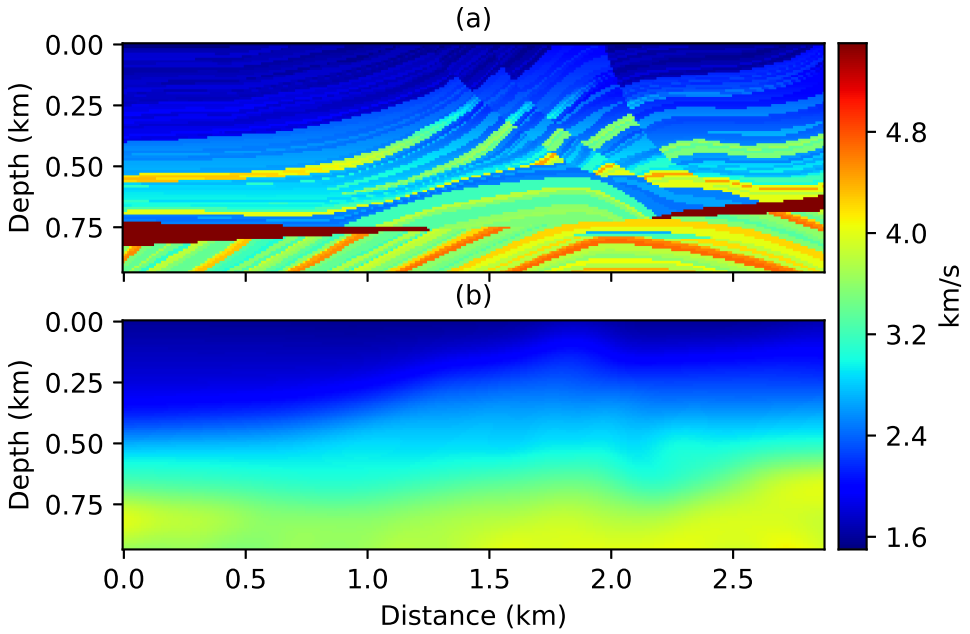

FIG. 16. The synthetic example of 2D Marmousi. (a) True velocity of Marmousi. (b) The initial velocity model of Marmousi for inversion.

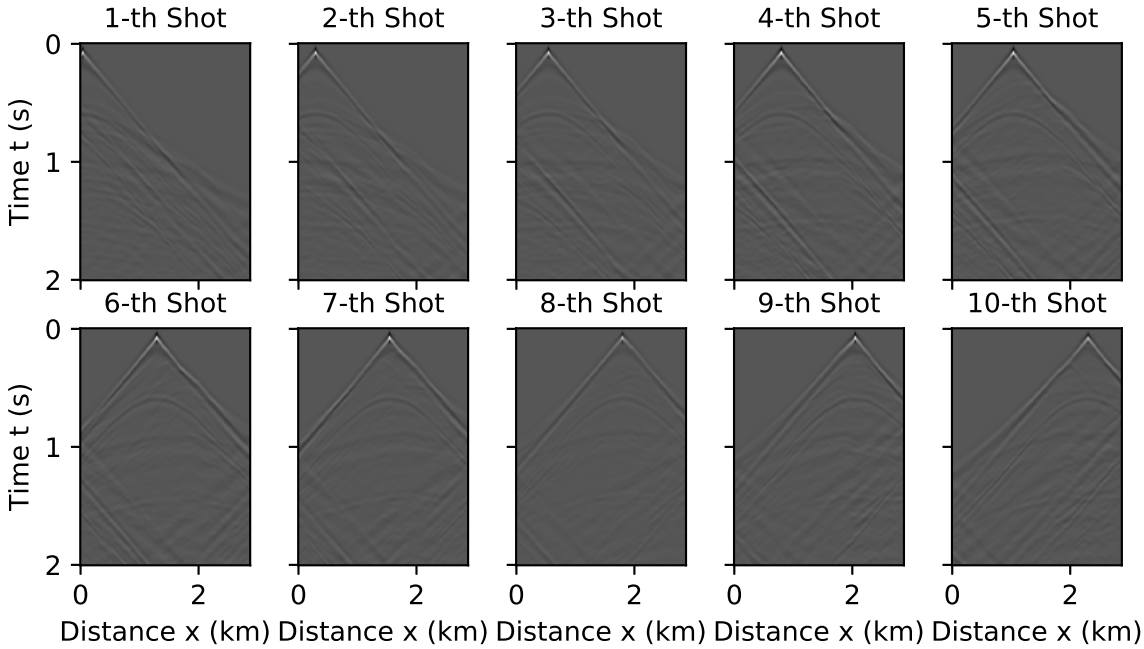

FIG. 17. The synthetic shot gathers generated using the true velocity of Marmousi shown in Figure 16a. The first 10 of 12 shots are plotted.

shown in Figure 16a.

The RNN inversion of 2D Marmousi model using non-linear CG method is also implemented and the predicted results at $[400, 800, 1420]th$ iterations are plotted In Figure 19. It is clear that, non-linear CG is not able to predict the details of Marmousi model with itera-

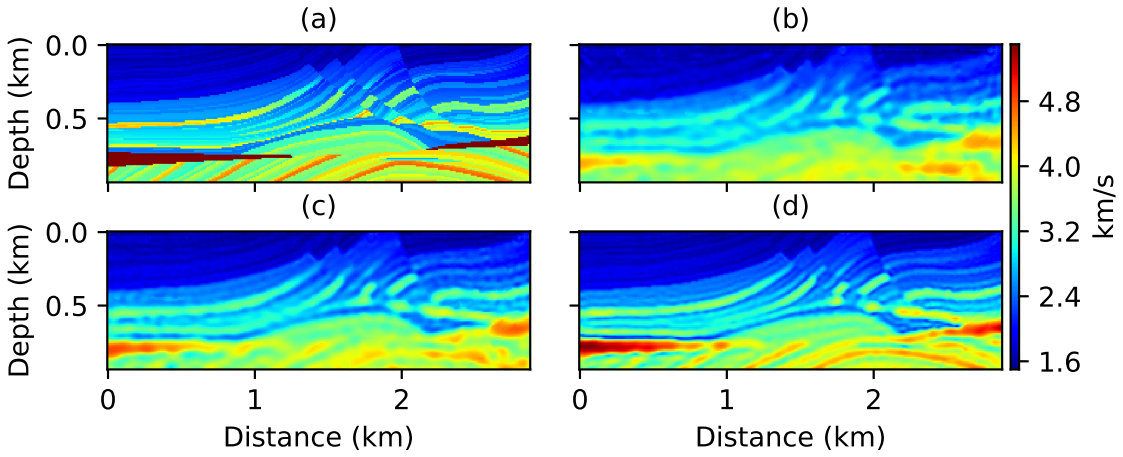

FIG. 18. The inversion result of Marmousi using Adam algorithm. (a) True Marmousi model. (b) Inversion at $25th$ iteration. (c) Inversion at $50th$ iteration. (d) Inversion at $100th$ iteration.

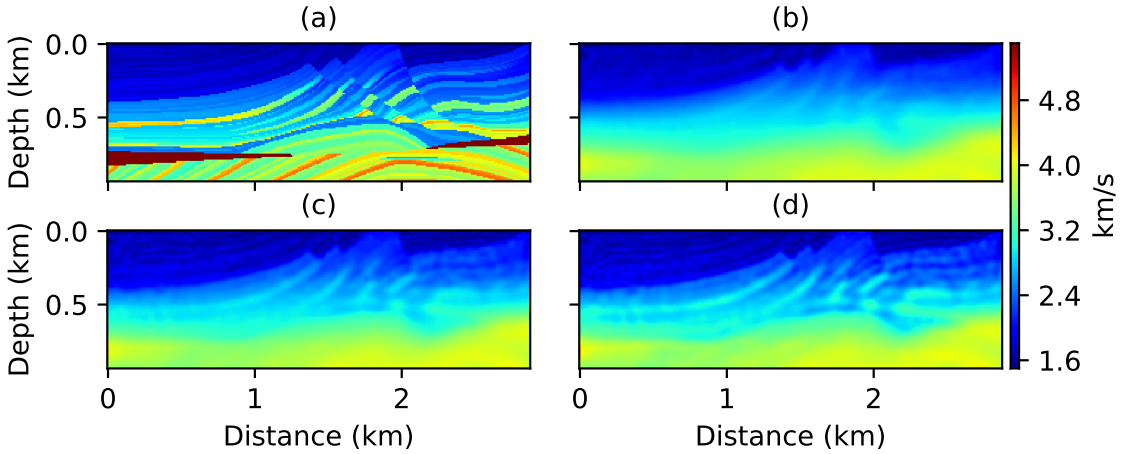

FIG. 19. The inversion result of Marmousi using CG algorithm. (a) True Marmousi model. (b) Inversion at $400th$ iteration. (c) Inversion at $800th$ iteration. (d) Inversion at $1420th$ iteration.

tion less than 400. Even after 800 iterations of decreasing the objective function, CG only provides limited information in the predicted model. Robust structures at shallow zones is visible after 1420 iterations, however, the predicted velocity values are still not even close to the true velocity model. Based on the analysis of the performance using non-linear CG on 1D depth-varying and 2D Marmousi model, it is to conclude that non-linear CG has the ability to smoothly converge the objective function, but requires a tremendous amounts of iterations and prohibitive computational cost time on a single PC.

The numerical analysis in previous section shows that, L-BFGS algorithm are able to do a great work on 1D inversion as Adam does, but with a slightly slower convergence speed. The RNN inversion using L-BFGS algorithm on the 2D Marmousi model at $[200, 600, 1000]th$ iterations are subplotted in Figure 20, respectively. Comparing to the inversion using non-linear CG, the predicted results with L-BFGS shows much faster convergence speed and detailed information of structures and layers at shallow zones can be detected after 600 iterations of reducing objective function. With 1000 iterations, partial

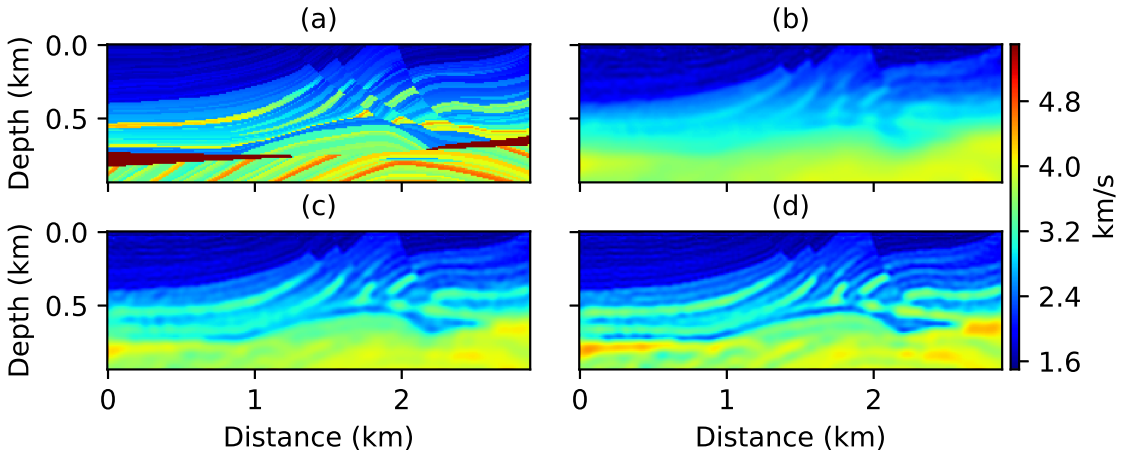

FIG. 20. The inversion result of Marmousi using L-BFGS algorithm. (a) True Marmousi model. (b) Inversion at $200th$ iteration. (c) Inversion at $600th$ iteration. (d) Inversion at $1000th$ iteration.

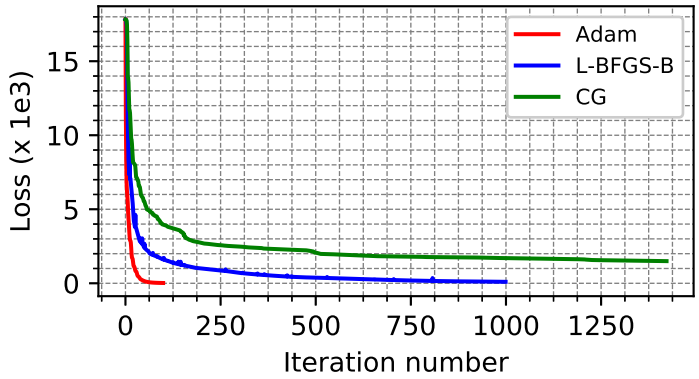

FIG. 21. The comparison of the loss values with Adam, CG, L-BFGS algorithms.

information of structures and layers at deep zones are also visible to determined. However, the computational cost of 1000 iterations using L-BFGS is very time-consuming and distances still exist between the predicted profile at $1000th$ iteration and the true Marmousi model.

To get a better view of the computational cost of each algorithm, we plot values of the objective function with respect to varying iteration number, shown in Figure 21. Remember, the calculating time of one-single iteration using these three algorithms is approximately identical to each other. Comparing to CG and L-BFGS, Adam has much faster and more stable convergence of the objective function. The comparison of best predicted velocity profiles using Adam (at $100th$ iteration), non-linear CG (at $1420th$ iteration), and L-BFGS (at $1000th$ iteration) algorithms at locations $x = [200, 1500, 2500]$m are also plotted in Figure 22. After 1420 iterations, non-linear CG is able to predict velocities in shallow zone and the correct trend of velocity changes in deep zones, however, it still requires a mass of iterations to recover the well-behaved prediction in deep zones. Figure 22b shows that L-BFGS algorithm achieves consistent prediction as Adam does at multi-fold areas (e.g, $x = 1500m$). However, for edge-areas of the model which are not covered by all shot

records, L-BFGS generates competitive results at shallow and intermediate-depth zones, but has significant lags at deep zones comparing to Adam method.

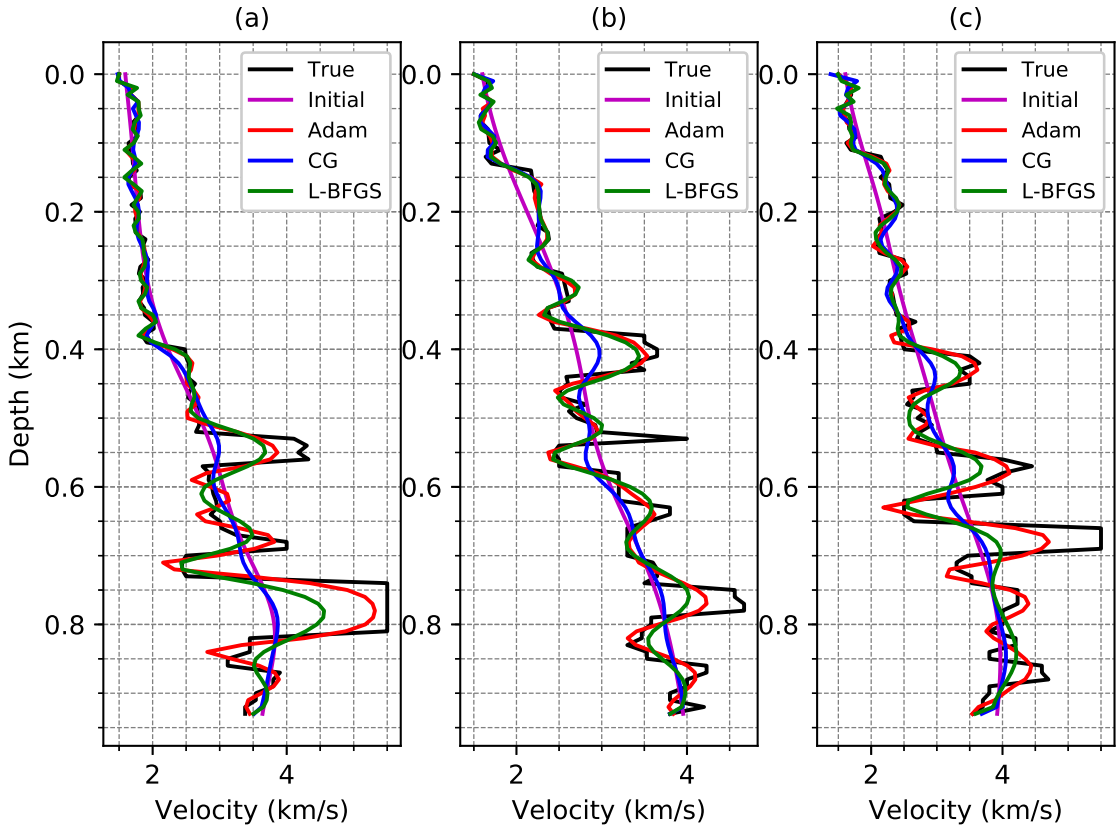

FIG. 22. Comparisons of velocity profile using Adam at $100th$ iteration, non-linear CG at $1420th$ iteration, and L-BFGS at $1000th$ iteration. (a) Velocity profiles at location $x = 200$m. (b) Velocity profiles at location $x = 1500$m. (c) Velocity profiles at location $x = 2500$m.

CONCLUSION

Deep learning has been tremendously improved and widely applied in different fields. To benefits from the well-developed deep learning community and open-source libraries, we proposed a self-designed recurrent neural networks which allows us to cast the forward modeling of seismic wave propagation into the forward propagation of RNN framework. As a consequence, the geophysical inversion problem is also turned into a training process of the presented RNN framework, where full wavefield information are involved. The derivation of the gradient formulation of the objective function with respect to the model parameters is illustrated in a deep learning perspective, and it further shows that the training process of the RNN is equivalent to a FWI problem. In other words, it is proven that FWI is also a specific machine learning case.

By delineating the geophysical inversion problem in sense of neural network training, the numerical analysis of hyperparameter tuning, mainly learning rate, is discussed using a depth-varying only velocity profile. The best appropriate range of learning rate for gradient-based algorithms on a geophysical velocity inversion problems are theoretically analyzed and experimentally investigated. Further, the comparison between different

gradient-based algorithms are also studied based on their best performance with appropriate learning rates. Beyond that, the comparison is expanded to the non-linear optimization algorithms, such as non-linear CG, L-BFGS, and TNC methods. The analysis shows that Adam is capable of converging the objective function as L-BFGS algorithm, which is much faster than non-linear CG, TNC, and the rest of gradient-based algorithms.

To fully examine their capacity, the RNN inversion is applied to 2D Marmousi model using Adam, CG, and L-BFGS methods. The results show that non-linear CG requires tremendous iterations and computational cost to predict the full information of structures and layers at both shallow and deep zones in Marmousi model, while L-BFGS has a better convergence speed of the objective function. However, to completely recover detailed information, L-BFGS also demands a relatively heavy computational cost. Comparing to CG and L-BFGS, the gradient-based Adam algorithm is able to converge the objective function much faster and predicted fully detailed velocity information at both shallow and deep zones well-matched to the true Marmousi model, by scaling the gradient with accumulating itself and its squared-norm tuned by hyperparameters.

ACKNOWLEDGMENTS

We thank the sponsors of CREWES for continued support. This work was funded by CREWES industrial sponsors and NSERC (Natural Science and Engineering Research Council of Canada) through the grant CRDPJ 461179-13.

APPENDIX A: DERIVATION OF THE GRADIENT IN THE RNN

The mathematical formulation of the objective function for geophysical inversion problems is written as,

$$J(\boldsymbol{v}) = \frac{1}{2n_s} \sum_{\mathbf{r_s}} \sum_{\mathbf{r_g}} \sum_{\mathbf{t}} (\mathbf{d_t} - \tilde{\mathbf{d}}_\mathbf{t})^2 = \frac{1}{2n_s} \sum_{\mathbf{r_s}} \sum_{\mathbf{r_g}} \sum_{\mathbf{t}} (\mathbf{d_t} - \boldsymbol{\delta}_{\mathbf{r_g}} \tilde{\mathbf{u}}_\mathbf{t})^2 \tag{A-1}$$

where $\mathbf{d_t}$ and $\tilde{\mathbf{d}}_\mathbf{t}$ represent the label (i.e., observed) and predicted data, respectively. $\mathbf{r_s}$ and $\mathbf{r_g}$ delineate the source and receiver coordinates. $\mathbf{t}$ represents the time vector. Receiver locations are denoted by $\boldsymbol{\delta}_{\mathbf{r_g}}$.

Using the chain rule, the partial derivative of the objective function over the trainable parameter, i.e., velocity, can be written as,

$$\frac{\partial J}{\partial \boldsymbol{v}} = \sum_{\mathbf{t}}^{\mathbf{T}} \left[\frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}} \right] \frac{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}}{\partial \boldsymbol{v}} \tag{A-2}$$

At the last time step ($t = T$), the partial derivative $[\partial J / \partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}]_{t=T}$ is calculated as

$$\left[\frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}}\right]_{t=T} = \frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{T}}} = -\frac{1}{n_s}\sum_{\mathbf{r_s}}\sum_{\mathbf{r_g}}(\mathbf{d_T} - \boldsymbol{\delta}_{\mathbf{r_g}}\tilde{\mathbf{u}}_{\mathbf{T}}) \tag{A-3}$$

When at the last second time step $t = T-1$, because of the dependency between $\tilde{\mathbf{u}}_{\mathbf{T-1}}$ and $\tilde{\mathbf{u}}_{\mathbf{T}}$, the partial derivative $[\partial J/\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}]_{t=T-1}$ is calculated by two separated terms based on the chain rule, shown as,

$$\left[\frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}}\right]_{t=T-1} = \left[\frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}}\right]_{t=T}\frac{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{T}}}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{T-1}}} + \frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{T-1}}} \tag{A-4}$$

For the time step where $0 < t < T-2$, the partial derivative $[\partial J/\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}]$ are determined by three dependency terms when the second-order finite-difference is considered for forward modeling, which is written as,

$$\left[\frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}}\right] = \left[\frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t+2}}}\right]\frac{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t+2}}}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}} + \left[\frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t+1}}}\right]\frac{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t+1}}}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}} + \frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}} \tag{A-5}$$

By considering the 2nd-order finite-difference forward modeling,

$$\tilde{\mathbf{u}}_{\mathbf{t+2}} = \boldsymbol{v}^2\Delta t^2(\nabla^2\tilde{\mathbf{u}}_{\mathbf{t+1}} - \boldsymbol{s}_{\boldsymbol{t+1}}) + 2\tilde{\mathbf{u}}_{\mathbf{t+1}} - \tilde{\mathbf{u}}_{\mathbf{t}} \tag{A-6}$$

The partial derivatives of $\mathbf{u_{t+2}}$ over $\mathbf{u_{t+1}}$, $\mathbf{u_t}$ and $\boldsymbol{v}$ can be expressed as,

$$\frac{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t+2}}}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t+1}}} = \boldsymbol{v}^2\Delta t^2\nabla^2 + 2 \tag{A-7a}$$

$$\frac{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t+2}}}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}} = -1 \tag{A-7b}$$

$$\frac{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t+2}}}{\partial \boldsymbol{v}} = 2\boldsymbol{v}\Delta t^2(\nabla^2\tilde{\mathbf{u}}_{\mathbf{t+1}} - \boldsymbol{s}_{\boldsymbol{t+1}}) \tag{A-7c}$$

Substitute equations (A-3) and (A-7a) into equation (A-4), we have,

$$\left[\frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}}\right]_{t=T-1} = -\frac{1}{n_s}\sum_{\mathbf{r_s}}\sum_{\mathbf{r_g}}[(\boldsymbol{v}^2\Delta t^2\nabla^2 + 2)\boldsymbol{\delta}\mathbf{d_T} + \boldsymbol{\delta}\mathbf{d_{T-1}}] \tag{A-8}$$

where $\boldsymbol{\delta}\mathbf{d_t}$ represents the residual between observed and predicted data at a single time step t, which is calculated as $\boldsymbol{\delta}\mathbf{d_t} = \mathbf{d_t} - \boldsymbol{\delta}_{\mathbf{r_g}}\tilde{\mathbf{u}}_{\mathbf{t}}$.

Reorganizing equation (A-5) by substituting equations (A-7a) and (A-7b),

$$\left[\frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}}\right] = \boldsymbol{v}^2\Delta t^2\left(\nabla^2\left[\frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t+1}}}\right] - \frac{1}{n_s\boldsymbol{v}^2\Delta t^2}\sum_{\mathbf{r_s}}\sum_{\mathbf{r_g}}\boldsymbol{\delta}\mathbf{d_t}\right) + 2\left[\frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t+1}}}\right] - \left[\frac{\partial J}{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t+2}}}\right] \tag{A-9}$$

Equation A-9 shows that the partial derivative of the objective function over predicted wavefield $[\partial J/\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}]$ is performed by propagating the scaled data residual in reversal time, which initial states of back propagation are indicated by equations A-3 and A-8.

With the wave equation, equation (A-7c) can be rewritten as,

$$\frac{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}}{\partial \boldsymbol{v}} = \frac{2\Delta t^2}{\boldsymbol{v}}\boldsymbol{v}^2(\nabla^2\tilde{\mathbf{u}}_{\mathbf{t+1}} - \boldsymbol{s}_{t+1}) \approx \frac{2\Delta t^2}{\boldsymbol{v}}\frac{\partial^2\tilde{\boldsymbol{u}}_{\boldsymbol{t}}}{\partial t^2} \tag{A-10}$$

Therefore, the gradient for updated model is obtained as,

$$\begin{aligned} g &= \frac{\partial J}{\partial v} \\ &= BP(-\frac{1}{n_s\boldsymbol{v}^2\Delta t^2}\sum_{\mathbf{r_s}}\sum_{\mathbf{r_g}}\boldsymbol{\delta}\mathbf{d_t})\frac{\partial \tilde{\boldsymbol{u}}_{\boldsymbol{t}}}{\partial \boldsymbol{v}} \\ &= BP(-\frac{1}{n_s\boldsymbol{v}^2\Delta t^2}\sum_{\mathbf{r_s}}\sum_{\mathbf{r_g}}\boldsymbol{\delta}\mathbf{d_t})\frac{2\Delta t^2}{\boldsymbol{v}}\frac{\partial^2\tilde{\boldsymbol{u}}_{\boldsymbol{t}}}{\partial t^2} \\ &\approx BP(-\frac{1}{n_s}\sum_{\mathbf{r_s}}\sum_{\mathbf{r_g}}\boldsymbol{\delta}\mathbf{d_t})\frac{2}{\boldsymbol{v}^3}\frac{\partial^2\tilde{\boldsymbol{u}}_{\boldsymbol{t}}}{\partial t^2} \end{aligned} \tag{A-11}$$

where BP indicates the back propagation of residuals.

As denoted by equation (A-11), the gradient in RNN's framework is calculated by the cross-correlation between second-order partial derivative of forward wavefield over time and the time-reversal wavefield using the residuals as the source, which is equivalent to the gradient achieved in the full waveform inversion (FWI). In other words, the FWI is also the deep learning process performed in one special type of RNN framework.

APPENDIX B: PSEUDO-CODES FOR MOMENTUM, ADAGRAD, RMSPROP, AND ADAM ALGORITHMS

To make this paper readable, in this section, the pseudo-codes of gradient-based algorithms, such as Momentum, Adagrad, RMSprop, and Adam, are provided. And to be consistent and to be clear about relationship of these methods, the variables and parameters are re-organized comparing to the published literatures.

Algorithm 1 Momentum (Qian, 1999): Gradient with momentum algorithm. Recommended setting for hyper-parameters: $\beta = 0.9$. All operations on vectors are element-wise. With β^k denote β to the power of k.

Require: α: Learning rate or step size.
Require: $\beta \in [0, 1)$: Exponential decay rates for the moment estimates.
Require: K: Maximum training step.
Require: $\boldsymbol{\delta v}_k$: Perturbation models achieved from gradient descent method.

Initial states for the moment parameter vector: $\boldsymbol{m}_0 = 0$.
Initializing training step $k = 0$.
while $\boldsymbol{v}_k$ not converged and $k \leq K$ **do**
    $k \leftarrow k + 1$
    $\boldsymbol{m}_k \leftarrow \beta \cdot \boldsymbol{m}_{k-1} + (1 - \beta) \cdot \boldsymbol{\delta v}_k$ (Update biased momentum)
    $\tilde{\boldsymbol{m}}_k \leftarrow \boldsymbol{m}_k / (1 - \beta^k)$ (Bias correction for the momentum)
    $\boldsymbol{v}_k \leftarrow \boldsymbol{v}_{k-1} - \alpha \cdot \tilde{\boldsymbol{m}}_k$ (Parameters updates)
end while

return $\boldsymbol{v}_k$

Algorithm 2 Adagrad (Duchi et al., 2011): Adaptive gradient algorithm. Recommended setting for $\epsilon = 10^{-8}$. All operations on vectors are element-wise.

Require: α: Learning rate or step size.
Require: $\beta \in [0, 1)$: Exponential decay rates for the moment estimates.
Require: K: Maximum training step.
Require: $\boldsymbol{\delta v}_k$: Perturbation models achieved from gradient descent method.

Initial states for diagonal matrix: $\boldsymbol{G}_0 = 0$, .
Initializing training step $k = 0$.
while $\boldsymbol{v}_k$ not converged and $k \leq K$ **do**
    $k \leftarrow k + 1$
    $\boldsymbol{G}_{k,ii} \leftarrow \boldsymbol{G}_{k-1,ii} + \boldsymbol{\delta v}_{k,i}^2$ (Update biased momentum)
    $\boldsymbol{v}_{k,i} \leftarrow \boldsymbol{v}_{k-1} - \frac{\alpha}{\sqrt{\boldsymbol{G}_{k,ii} + \epsilon}} \cdot \boldsymbol{\delta v}_{k,i}$ (Parameters updates)
end while

return $\boldsymbol{v}_k$

Algorithm 3 RMSprop (Hinton et al., 2012b): Root-Mean-Squared gradients algorithm. Recommended setting for hyper-parameters: $\beta = 0.9$, and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β^k denote β to the power of k.

Require: α: Learning rate or step size.
Require: $\beta \in [0, 1)$: Exponential decay rates for the moment estimates.
Require: K: Maximum training step.
Require: $\boldsymbol{\delta v}_k$: Perturbation models achieved from gradient descent method.
Initial states for the moment parameter vector: $\boldsymbol{r}_0 = 0$.
Initializing training step $k = 0$.
while $\boldsymbol{v}_k$ not converged and $k \leq K$ **do**
    $k \leftarrow k + 1$
    $\boldsymbol{r}_k \leftarrow \beta \cdot \boldsymbol{r}_{k-1} + (1 - \beta) \cdot \boldsymbol{\delta v}_k^2$ (Update biased momentum)
    $\tilde{\boldsymbol{r}}_k \leftarrow \boldsymbol{r}_k / (1 - \beta^k)$ (Bias correction for the momentum)
    $\boldsymbol{v}_k \leftarrow \boldsymbol{v}_{k-1} - \alpha \cdot \boldsymbol{\delta v}_k / (\sqrt{\tilde{\boldsymbol{r}}_k} + \epsilon)$ (Parameters updates)
end while
return $\boldsymbol{v}_k$

Algorithm 4 Adam (Kingma and Ba, 2014): Adaptive momentum algorithm. Recommended setting for hyper-parameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^k and β_2^k denote β_1 and β_2 to the power of k.

Require: α: Learning rate or step size.
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates.
Require: K: Maximum training step.
Require: $\boldsymbol{\delta v}_k$: Perturbation models achieved from gradient descent method.
Initial states for the moment parameter vectors: $\boldsymbol{m}_0 = 0$, $\boldsymbol{r}_0 = 0$.
Initializing training step $k = 0$.
while $\boldsymbol{v}_k$ not converged and $k \leq K$ **do**
    $k \leftarrow k + 1$
    $\boldsymbol{m}_k \leftarrow \beta_1 \cdot \boldsymbol{m}_{k-1} + (1 - \beta_1) \cdot \boldsymbol{\delta v}_k$ (Update biased 1st momentum)
    $\boldsymbol{r}_k \leftarrow \beta_2 \cdot \boldsymbol{r}_{k-1} + (1 - \beta_2) \cdot \boldsymbol{\delta v}_k^2$ (Update biased 2nd momentum)
    $\tilde{\boldsymbol{m}}_k \leftarrow \boldsymbol{m}_k / (1 - \beta_1^k)$ (Bias correction for 1st momentum)
    $\tilde{\boldsymbol{r}}_k \leftarrow \boldsymbol{r}_k / (1 - \beta_2^k)$ (Bias correction for 2nd momentum)
    $\boldsymbol{v}_k \leftarrow \boldsymbol{v}_{k-1} - \alpha \cdot \tilde{\boldsymbol{m}}_k / (\sqrt{\tilde{\boldsymbol{r}}_k} + \epsilon)$ (Parameters updates)
end while
return $\boldsymbol{v}_k$

REFERENCES

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

Broyden, C. G., 1970, The convergence of a class of double-rank minimization algorithms: 2. the new algorithm: IMA journal of applied mathematics, **6**, No. 3, 222–231.

Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C., 1995, A limited memory algorithm for bound constrained optimization: SIAM Journal on Scientific Computing, **16**, No. 5, 1190–1208.

Carcione, J. M., Herman, G. C., and Ten Kroode, A., 2002, Seismic modeling: Geophysics, **67**, No. 4, 1304–1325.

Cireşan, D., Meier, U., and Schmidhuber, J., 2012, Multi-column deep neural networks for image classification: arXiv preprint arXiv:1202.2745.

Collobert, R., and Weston, J., 2008, A unified architecture for natural language processing: Deep neural networks with multitask learning, *in* Proceedings of the 25th international conference on Machine learning, ACM, 160–167.

Dechter, R., 1986, Learning while searching in constraint-satisfaction problems: University of California, Computer Science Department, Cognitive Systems Laboratory.

Defferrard, M., Bresson, X., and Vandergheynst, P., 2016, Convolutional neural networks on graphs with fast localized spectral filtering, *in* Advances in Neural Information Processing Systems, 3844–3852.

Duchi, J., Hazan, E., and Singer, Y., 2011, Adaptive subgradient methods for online learning and stochastic optimization: Journal of Machine Learning Research, **12**, No. Jul, 2121–2159.

Etgen, J. T., and Brandsberg-Dahl, S., 2009, The pseudo-analytical method: Application of pseudo-laplacians to acoustic and acoustic anisotropic wave propagation, *in* SEG Technical Program Expanded Abstracts 2009, Society of Exploration Geophysicists, 2552–2556.

Eymard, R., Gallouët, T., and Herbin, R., 2000, Finite volume methods: Handbook of numerical analysis, **7**, 713–1018.

Faccioli, E., Maggio, F., Paolucci, R., and Quarteroni, A., 1997, 2d and 3d elastic wave propagation by a pseudo-spectral domain decomposition method: Journal of seismology, **1**, No. 3, 237–251.

Fletcher, R., 1970, A new approach to variable metric algorithms: The computer journal, **13**, No. 3, 317–322.

Fletcher, R., and Reeves, C. M., 1964, Function minimization by conjugate gradients: The computer journal, **7**, No. 2, 149–154.

Glinsky-Olivier, N., Jemaa, M. B., Virieux, J., and Piperno, S., 2006, 2d seismic wave propagation by a finite-volume method, *in* 68th EAGE Conference and Exhibition incorporating SPE EUROPEC 2006.

Goldfarb, D., 1970, A family of variable-metric methods derived by variational means: Mathematics of computation, **24**, No. 109, 23–26.

Goodfellow, I., Bengio, Y., and Courville, A., 2016, Deep Learning: MIT Press, `http://www.deeplearningbook.org`.

Graves, A., Jaitly, N., and Mohamed, A.-r., 2013a, Hybrid speech recognition with deep bidirectional lstm, *in* Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on, IEEE, 273–278.

Graves, A., Mohamed, A.-r., and Hinton, G., 2013b, Speech recognition with deep recurrent neural networks, *in* Acoustics, speech and signal processing (icassp), 2013 ieee international conference on, IEEE, 6645–6649.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N. et al., 2012a, Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups: IEEE Signal processing magazine, **29**, No. 6, 82–97.

Hinton, G., Srivastava, N., and Swersky, K., 2012b, Neural networks for machine learning lecture 6a overview of mini-batch gradient descent: Cited on, 14.

Hinton, G. E., Osindero, S., and Teh, Y.-W., 2006, A fast learning algorithm for deep belief nets: Neural computation, **18**, No. 7, 1527–1554.

Kingma, D. P., and Ba, J., 2014, Adam: A method for stochastic optimization: arXiv preprint arXiv:1412.6980.

Lailly, P., and Bednar, J., 1983, The seismic inverse problem as a sequence of before stack migrations, *in* Conference on inverse scattering: theory and application, Siam Philadelphia, PA, 206–220.

Li, J., Innanen, K. A., and Tao, G., 2018, A 3d pseudo-spectral method for sh wave simulation in heterogeneous media, *in* SEG Technical Program Expanded Abstracts 2018, Society of Exploration Geophysicists, 4005–4009.

Li, X., and Wu, X., 2015, Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition, *in* Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on, IEEE, 4520–4524.

Liu, Q., and Tromp, J., 2006, Finite-frequency kernels based on adjoint methods: Bulletin of the Seismological Society of America, **96**, No. 6, 2383–2397.

Marfurt, K. J., 1984, Accuracy of finite-difference and finite-element modeling of the scalar and elastic wave equations: Geophysics, **49**, No. 5, 533–549.

McGillivray, P. R., and Oldenburg, D., 1990, Methods for calculating fréchet derivatives and sensitivities for the non-linear inverse problem: A comparative study 1: Geophysical prospecting, **38**, No. 5, 499–524.

Miljanovic, M., 2012, Comparative analysis of recurrent and finite impulse response neural networks in time series prediction: Indian Journal of Computer Science and Engineering, 180–191.

Morales, J. L., and Nocedal, J., 2011, Remark on algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound constrained optimization: ACM Transactions on Mathematical Software (TOMS), **38**, No. 1, 7.

Nocedal, J., and Wright, S. J., 2006, Conjugate gradient methods: Numerical optimization, 101–134.

Plessix, R.-E., 2006, A review of the adjoint-state method for computing the gradient of a functional with geophysical applications: Geophysical Journal International, **167**, No. 2, 495–503.

Qian, N., 1999, On the momentum term in gradient descent learning algorithms: Neural networks, **12**, No. 1, 145–151.

Robinson, A. J., Cook, G. D., Ellis, D. P., Fosler-Lussier, E., Renals, S., and Williams, D., 2002, Connectionist speech recognition of broadcast news: Speech Communication, **37**, No. 1-2, 27–45.

Sak, H., Senior, A., and Beaufays, F., 2014, Long short-term memory recurrent neural network architectures for large scale acoustic modeling, *in* Fifteenth annual conference of the international speech communication association.

Schmidhuber, J., 2015, Deep learning in neural networks: An overview: Neural networks, **61**, 85–117.

Shanno, D. F., 1970, Conditioning of quasi-newton methods for function minimization: Mathematics of computation, **24**, No. 111, 647–656.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. et al., 2016, Mastering the game of go with deep neural networks and tree search: nature, **529**, No. 7587, 484.

Smith, G. D., 1985, Numerical solution of partial differential equations: finite difference methods: Oxford university press.

Tarantola, A., 1984, Inversion of seismic reflection data in the acoustic approximation: Geophysics, **49**, No. 8, 1259–1266.

Thulasiraman, K., and Swamy, M. N., 2011, Graphs: theory and algorithms: John Wiley & Sons.

Virieux, J., 1986, P-sv wave propagation in heterogeneous media: Velocity-stress finite-difference method: Geophysics, **51**, No. 4, 889–901.

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

Yang, P., Gao, J., and Wang, B., 2015, A graphics processing unit implementation of time-domain full-waveform inversion: Geophysics, **80**, No. 3, F31–F39.

Yedlin, M. J., and Van Vorst, D., 2010, Tutorial on the continuous and discrete adjoint state method and basic implementation.

Zeiler, M. D., 2012, Adadelta: an adaptive learning rate method: arXiv preprint arXiv:1212.5701.

Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J., 1997, Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization: ACM Transactions on Mathematical Software (TOMS), **23**, No. 4, 550–560.