Inversion with the Born approximation in a deep learning framework

Zhan Niu, Jian Sun, Daniel O. Trad

ABSTRACT

Least squares reverse time migration (LSRTM) is an important technique that is starting to be used in the industry. LSRTM is closely related to Full-waveform Inversion (FWI) but instead of seeking for an optimal velocity model, it searches for an optimal reflectivity. Machine learning, on the other hand, has gained attention in the geophysics community and has become one of the most booming subjects in computer science. Various tools and methodology have been developed in the last few years and geophysicists have been finding applications by using these tools to solve more efficiently or with better quality long standing processing, imaging and interpretation problems. In this report, we first introduce an implementation of the Born modelling using the recurrent neural network (RNN) and second, we perform an inversion of the model by training the RNN with generated data. The inversion process can be proven to be same as LSRTM. The performance of different optimizers is compared and discussed. We conclude that the ADAM optimizer is the most stable and time efficient for this method.

INTRODUCTION

Most machine learning algorithms (ML) treat problems from an statistical perspective. Like linear regression for example, they extract features (model parameters) from the given data and learn how to correlate the selected features to the data (Goodfellow et al., 2016). Therefore, the efficiency of these algorithms is highly dependent on (1) how the features are chosen and (2) what role these features play in the learning process. This role depends on the structure of the neural network (NN). The NN structure can be thought of as the way we inject our knowledge into the algorithm to make the network to learn from the data. Even the most fundamental fully-connected neural network contains injected knowledge—it needs the number of layers and the number of nodes to be predefined, which is related to how many orders of non-linearity need to be simulated. The convolution neural network (CNN), which is popular with image recognition/segmentation, injects the idea that a feature at a given point only depends on its nearby points (Fukushima, 1979; LeCun et al., 2015). This assumption greatly reduces the burden on the learning process compared to the fully-connected neural networks. The recurrent neural network (RNN) feeds in the knowledge that the output at a current state depends on the features at the current state and the previous states (Lipton et al., 2015). This characteristic makes the network time or sequence relevant and makes RNN a perfect match for a complex job like word recognition and natural language processing. As a general rule, the more knowledge is feed into the neural network (structure), the easier it can be trained.

Most studies applying machine learning to geophysics treat the forward problem as a black box and select the velocity as a feature (Richardson, 2018; Moseley et al., 2018). The black box idea solves the problem statistically, which follows the classical machine learning philosophy, but it ignores theoretical knowledge that has been well studied in geo-

physics (e.g. the wave equation and scattering theory). With the neglect of those crucial theories, the neural network will spend too much energy finding approximations to the theories by itself. The approximations are usually poor and highly dependent on the problem trying to solve. Some recent works in geophysics proposed that we shall inject our knowledge to the machine learning algorithm and only treat part of it to be a numerical problem (Karpatne et al., 2017). By following a similar idea, in this work we incorporated the Born approximation into the structure of the RNN.

In this report, we attempt to add the wave propagation knowledge to an RNN. The RNN takes a background velocity and the source as input features. We designed the network structure from scratch to make the velocity perturbation to be the hidden parameter. The output of the RNN is a shot record. We implemented the RNN based on the APIs in TensorFlow, tested popular machine learning optimizers and discuss their performances.

THEORY

Forward modelling with the Born approximation

Consider a wavefield $\boldsymbol{p}_0(\boldsymbol{x},t)$ propagating in a background velocity $\boldsymbol{v}_0(\boldsymbol{x})$ with a source $\boldsymbol{f}(\boldsymbol{x},t)$. This wavefield obeys the wave equation

$$\left(\frac{1}{\boldsymbol{v}_0^2}\frac{\partial^2}{\partial t^2}-\nabla^2\right)\boldsymbol{p}_0=\boldsymbol{f} \tag{1}$$

Let us consider another wavefield $\boldsymbol{p}(\boldsymbol{x},t)=\boldsymbol{p}_0(\boldsymbol{x},t)+\delta\boldsymbol{p}(\boldsymbol{x},t)$ propagating in a velocity media $\boldsymbol{v}(\boldsymbol{x})=\boldsymbol{v}_0(\boldsymbol{x})+\delta\boldsymbol{v}(\boldsymbol{x})$ with the same source, where $v(\boldsymbol{x})$ is a velocity that differs from $\boldsymbol{v}_0(\boldsymbol{x})$ by $\delta\boldsymbol{v}(\boldsymbol{x})$ and $p(\boldsymbol{x},t)$ is the corresponding wavefield with respect to $\boldsymbol{v}(\boldsymbol{x})$. The wavefield is hence governed by

$$\left(\frac{1}{(\boldsymbol{v}_0+\delta\boldsymbol{v})^2}\frac{\partial^2}{\partial t^2}-\nabla^2\right)(\boldsymbol{p}_0+\delta\boldsymbol{p})=\boldsymbol{f} \tag{2}$$

If it is assumed that $\frac{\delta\boldsymbol{v}}{\boldsymbol{v}_0}\to 0$, then

$$\frac{1}{(\boldsymbol{v}_0+\delta\boldsymbol{v})^2}=\frac{1}{\boldsymbol{v}_0^2}\left(1+\frac{\delta\boldsymbol{v}}{\boldsymbol{v}_0}\right)^{-2}\approx\frac{1}{\boldsymbol{v}_0^2}\left(1-2\frac{\delta\boldsymbol{v}}{\boldsymbol{v}_0}\right) \tag{3}$$

Replacing Equation 1 in Equation 2

$$\left(\frac{1}{\boldsymbol{v}_0^2}\frac{\partial^2}{\partial t^2}-\nabla^2\right)\delta\boldsymbol{p}=\frac{2\delta\boldsymbol{v}}{\boldsymbol{v}_0^3}\frac{\partial^2}{\partial t^2}(\boldsymbol{p}_0+\delta\boldsymbol{p}) \tag{4}$$

If we apply the Born approximation assuming the second time derivative of $\delta\boldsymbol{p}$ is negligible, we can write two wave equations for the background and perturbed wavefield as follows:

$$\left(\frac{1}{\boldsymbol{v}_0^2}\frac{\partial^2}{\partial t^2}-\nabla^2\right)\boldsymbol{p}_0=\boldsymbol{f} \tag{5a}$$

$$\left(\frac{1}{\boldsymbol{v}_0^2}\frac{\partial^2}{\partial t^2}-\nabla^2\right)\delta\boldsymbol{p}=\frac{1}{\boldsymbol{v}_0^2}\cdot\boldsymbol{m}\cdot\frac{\partial^2}{\partial t^2}\boldsymbol{p}_0 \tag{5b}$$

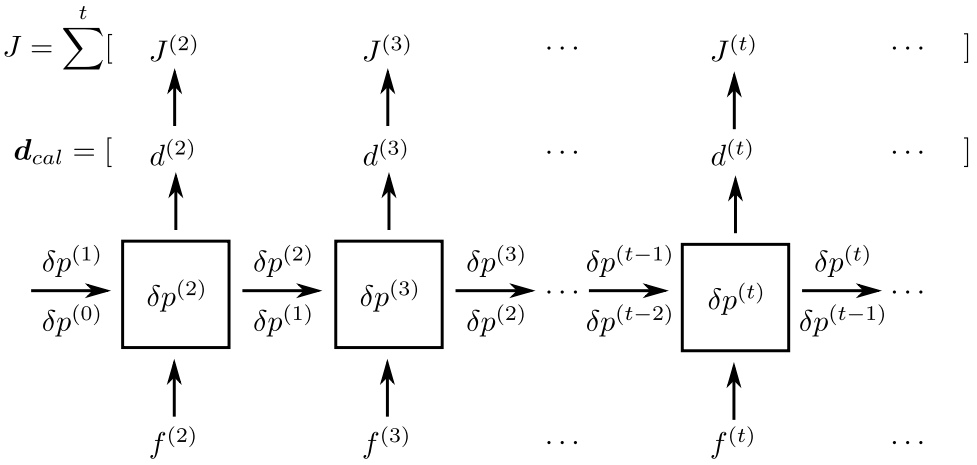

FIG. 1. The diagram of the RNN structure. The black boxes are neural cells which take the source and two previous perturbation wavefields to compute the next perturbation wavefields. The output of each cell is the shot record at a given time and the most recent two wavefields will be passed to the next cell.

where $\boldsymbol{m}$ is the model defined as velocity perturbation ($\frac{2\delta \boldsymbol{v}}{\boldsymbol{v}_0}$). The perturbation wavefield can be considered as the wavefield of a source that is the zero-lag cross-correlation of the model and the second time derivative of the background wavefield. With Equation 5a and 5b, we can calculate the perturbation wavefield ($\delta \boldsymbol{p}$) given a velocity perturbation $\boldsymbol{m}$ from $\boldsymbol{v}_0$.

The implementation using TensorFlow

Base on the fact that the wave propagation is a function of time, the most suitable neural network framework for simulating the wavefield is the recurrent neural network (RNN) (Lipton et al., 2015), which already contains time structures. The RNN can be built with the help of TensorFlow. TensorFlow (Abadi et al., 2015) brings the power of shared memory parallel computing and GPU acceleration. Moreover, TensorFlow has an API with which can we systematically compare different optimization methods. Figure 1 shows the RNN architecture used for this report.

As shown in Figure 1, each cell refers to a series of calculations to compute the perturbed wavefield. We decided to use the 3-point-centred finite difference expansion for the 2nd order time derivative since it is accurate enough for small time steps. In the space domain, we also chose a 2nd order finite difference for prototype development, which can be improved to a more accurate of approximation later on. The 2nd order approximation in the space is usually not adequate but is fast and stable. After discretization, Equation 5a and 5b can be rearranged to

$$\boldsymbol{p}_0^{(t+1)} - 2\boldsymbol{p}_0^{(t)} + \boldsymbol{p}_0^{(t-1)} = \boldsymbol{v}_0^2 \Delta t^2 \left(\nabla^2 \boldsymbol{p}_0^{(t)} + \boldsymbol{f} \right) \tag{6a}$$

$$\delta \boldsymbol{p}^{(t+1)} - 2\delta \boldsymbol{p}^{(t)} + \delta \boldsymbol{p}^{(t-1)} = \boldsymbol{v}_0^2 \Delta t^2 \left(\nabla^2 \delta \boldsymbol{p}^{(t)} + \frac{m}{{\boldsymbol{v}_0}^2} \left(\nabla^2 \boldsymbol{p}_0^{(t)} + \boldsymbol{f} \right) \right) \tag{6b}$$

where the superscript (t) denote a variable at the tth time step. Note that the time derivate $(\frac{\partial^2}{\partial t^2}\boldsymbol{p}_0)$ on the right hand side of Equation 5b is replaced by the equality in Equation 5a. Instead, in Equations 6a and 6b, the wavefields at the current time step can be calculated if the previous two wavefields are known or given. With the wavefield at the current cell, $d^{(t)}$ can be extracted from the wavefields and the cost $J^{(t)}$ can be calculated. Then the wavefields at (t) and $(t-1)$ will be forward to the next cell to go through a similar process until it reaches the maximum time step. The shot record ($\boldsymbol{d}_{cal}$) can be formed by concatenate $d^{(t)}$ from each cell and the cost J can also be computed if a measured data ($\boldsymbol{D}$) is given.

The gradient update and optimization

In the previous section, we introduce the procedure to use an RNN to perform forward modelling. Neural networks problems are essentially optimization problems. If we let a model parameter to be unknown and feed the generated data by the forward modelling, the model can be solved by "training" of the neural network.

The derivation of the gradient follows a similar idea as described by Richardson (2018). The cost function can be defined as

$$J = \frac{1}{2n_s}\sum_{\boldsymbol{x}_s}\|\boldsymbol{D}-\boldsymbol{d}_{cal}\|^2 = \frac{1}{2n_s}\sum_{\boldsymbol{x}_s}\sum_{\boldsymbol{x}_r}\sum_{t}(\boldsymbol{D}-\boldsymbol{d}_{cal})^2 = \frac{1}{2n_s}\sum_{\boldsymbol{x}_s}\sum_{\boldsymbol{x}_r}\sum_{t}\boldsymbol{r}^2, \quad (7)$$

where $\boldsymbol{D}$ stands for the observed data and $\boldsymbol{d}_{cal}$ is the prediction calculated by the Born modelling. $\boldsymbol{r}$ refers to the data residual. Since the error was summed through out the time, the gradient is the sum of all gradients at each time step as well:

$$\boldsymbol{g}_i = \sum_{t=0}^{T}\boldsymbol{g}_i^{(t)} = \sum_{t=0}^{T}\frac{\partial J^{(t)}}{\partial \boldsymbol{m}}. \quad (8)$$

The gradients at each time step can be proved to be the dot product of the time-reverse-propagated wavefield of the data residual ($\boldsymbol{r}$) and the 2nd time derivative of the time-forward-propagation of the background wavefield ($\boldsymbol{p}_0$), i.e.

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{m}} = \frac{1}{{\boldsymbol{v}_0}^2}\boldsymbol{B}(\boldsymbol{r},\boldsymbol{x},T-t)\cdot\frac{\partial^2}{\partial t^2}\boldsymbol{p}_0(f,\boldsymbol{x},t) \quad (9)$$

(proof is developed in the Appendix). Therefore, the optimization problem for the inverse of Born modelling can be treated in the same manner as a LSRTM inversion.

Optimization by adaptive moment estimation (ADAM)

For each iteration (called "epochs" in ML) of the training step, one can update the model using gradient descent as following:

$$\boldsymbol{m}_i = \boldsymbol{m}_{i-1} - \alpha\boldsymbol{g}_i \quad (10)$$

where α is the learning rate that is usually smaller than 1. However, the negative direction of the gradient is not necessarily the direction towards the local minimum. The traditional

gradient descent method is not costly to calculate at each iteration, but usually takes a zigzag path to the optimal solution. ADAM is a first-order optimization method which is able to suppress the oscillations that commonly appear in the gradient descent method. Similar to conjugate gradients, ADAM chooses a more optimal path based on previous updates. This optimization method combines the advantages of momentum and RMSprop (Kingma and Ba, 2014).

ADAM needs three (hyper-) parameters to be set manually: 1) α refers to the step length of the gradient update, which is similar what is used in other gradient methods; 2) β_1 and 3) β_2 are extra parameters to control how much the new gradient is related to previous gradients. These parameters are used to calculate two momentum terms, which are the accumulative sum of the first order and second order of the gradient in previous iterations. These momentums are defined as

$$\boldsymbol{v} \leftarrow \beta_1 \boldsymbol{v} + (1-\beta_1)\cdot \boldsymbol{g} \tag{11a}$$

$$\boldsymbol{s} \leftarrow \beta_2 \boldsymbol{s} + (1-\beta_2)\cdot \boldsymbol{g}^2. \tag{11b}$$

Instead of the model update described in Equation 10, ADAM optimizer updates the model in the following way:

$$\boldsymbol{m}_i = \boldsymbol{m}_{i-1} - \alpha \cdot \frac{\hat{\boldsymbol{v}}}{\sqrt{\hat{\boldsymbol{s}}}+\epsilon} \tag{12}$$

where $\hat{\boldsymbol{v}}$ and $\hat{\boldsymbol{s}}$ are $\boldsymbol{v}$ and $\boldsymbol{s}$ normalized by $(1-{\beta_1}^i)$ and $(1-{\beta_2}^i)$, respectively. ϵ is a regularization coefficient to avoid the division by zero. The pseudo-code provided by Kingma and Ba (2014) is shown below as Algorithm 1.

Algorithm 1 The ADAM optimization

$\boldsymbol{v}_0 \leftarrow 0$
$\boldsymbol{s}_0 \leftarrow 0$
$i \leftarrow 0$
while $i < niter$ **do**
  $i \leftarrow i+1$
  Calculate $\boldsymbol{g}_i$
  $\boldsymbol{v}_i \leftarrow \beta_1 \boldsymbol{v}_{i-1} + (1-\beta_1)\boldsymbol{g}_i$
  $\boldsymbol{s}_i \leftarrow \beta_2 \boldsymbol{s}_{i-1} + (1-\beta_2){\boldsymbol{g}_i}^2$
  $\hat{\boldsymbol{v}}_i \leftarrow \frac{\boldsymbol{v}_i}{1-{\beta_1}^i}$
  $\hat{\boldsymbol{s}}_i \leftarrow \frac{\boldsymbol{s}_i}{1-{\beta_2}^i}$
  $\boldsymbol{m}_i = \boldsymbol{m}_{i-1} - \alpha \cdot \frac{\hat{\boldsymbol{v}}_i}{\sqrt{\hat{\boldsymbol{s}}_i}+\epsilon}$

Generally speaking, the numerator of the update $\hat{v}$ can be interpreted as the weighted sum of gradients through iterations. Therefore the oscillation will be cancelled in terms of the vector sum. The denominator $\sqrt{\hat{s}}$ can be thought as the weighted root-sum-square of the gradients. For the very first iteration of ADAM, the update will be a scale of alpha and the information of the gradient will be added later on. From the flow of the ADAM optimization, one can notice that the absolute value of the parameter update will never be greater than the stepsize α. This means the ADAM optimizer will still perform small updates even if the optimal update should be large. This is because $\hat{\boldsymbol{v}} < \sqrt{\hat{\boldsymbol{s}}}$ holds as long as

gradients at each iteration are not always in the same direction. This characteristic prevents overshooting of the gradient. In our problem, the parameter to be solved is usually smaller than 1 (unless the initial guess for the velocity is very inaccurate), so the updates should be smaller than 1 as well. This is similar to classical machine learning problems, where the hidden parameters are usually small numbers. Therefore, we can start by testing different α values from 0.001 to 0.1, which works well in most machine learning algorithms. However, for problems that need greater updates, one may need a step size that is greater than the values recommended by machine learning.

The Fletcher-Reeves method

The Fletcher-Reeves method (FR) is a non-linear adaptation of the traditional linear conjugate gradient method (Wright and Nocedal, 1999). The traditional linear conjugate gradient method is designed for a quadratic cost function with respect to the model parameters. Although the Born modelling is a linear method and applying non-linearity seems to be unnecessary, the cost function may not be perfectly quadratic with respect to the model. Therefore, we decided to test this method by using the FR optimizer in the Scipy module of Python, which was implemented following Wright and Nocedal (1999). The pseudo-code is shown as Algorithm 2.

Algorithm 2 The Fletcher-Reeves method

Require: The initial guess of model $\boldsymbol{m}_0$ (zeros)
  Compute $J_0 = f(\boldsymbol{m}_0), \nabla J_0 = \frac{\partial J}{\partial \boldsymbol{m}}(\boldsymbol{m}_0)$
  Set $\boldsymbol{p}_0 = -\nabla J_0$
  $i \leftarrow 0$
  while $i < niter$ and $\nabla J_i \neq 0$ **do**
    Compute α_i and set $\boldsymbol{m}_{i+1} = \boldsymbol{m}_i + \alpha_i \boldsymbol{p}_i$
    Evaluate ∇J_{i+1}
    $\beta_{k+1} \leftarrow \frac{\langle \nabla J_{i+1}, \nabla J_{i+1} \rangle}{\langle \nabla J_i, \nabla J_i \rangle}$ ($\langle \cdot, \cdot \rangle$ refers to inner product)
    $\boldsymbol{p}_{i+1} \leftarrow -\nabla J_{i+1} + \beta_{k+1} \boldsymbol{p}_k$
    $i \leftarrow i + 1$

SYNTHETIC DATA EXAMPLES

The modelling results

We designed a simple scattering model to test the modelling accuracy using the Born approximation. The model is shown in Figure 2, which has a dimension of 101×101 cells with a grid spacing of $10\,\mathrm{m}$. For both the finite difference modelling and the Born modelling , 11 shots are fired at the indexes 1, 11, 21, 31, 41, 51, 61, 71, 81, 91 and 101 at the surface and the receivers are placed at each cell with the same depth. The receivers record the first $1\,\mathrm{s}$ with time step of $1\,\mathrm{ms}$. The injected source was a $15\,\mathrm{Hz}$ Ricker wavelet. Figures 3 and 4 are the corresponding synthetic wavefields, showing that finite difference and Born modelling produce similar results for this case.

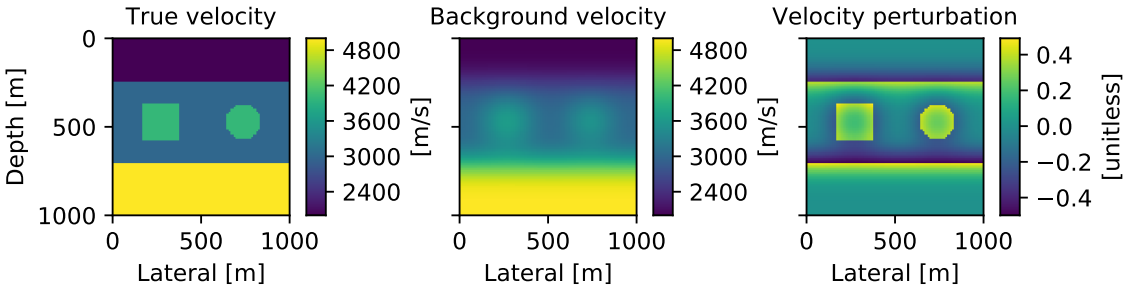

FIG. 2. The scattering model. The true model was used for the finite difference modelling (Figure 3) while the background and perturbed model were used for the Born modelling method (Figure 4).

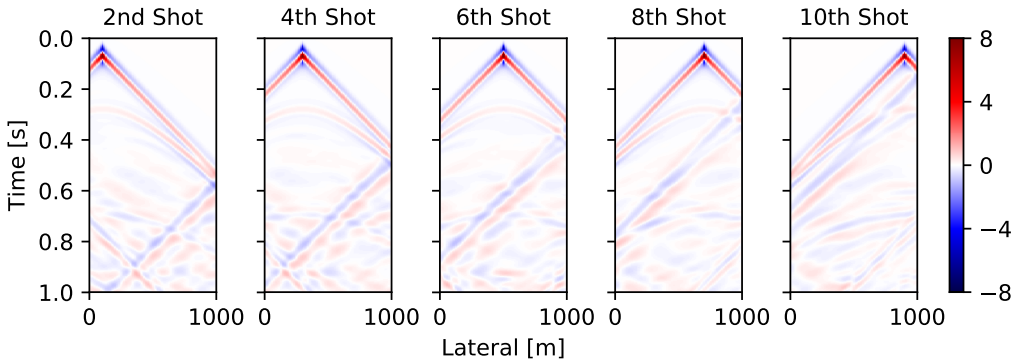

FIG. 3. A shot record calculated by the finite difference modelling method.

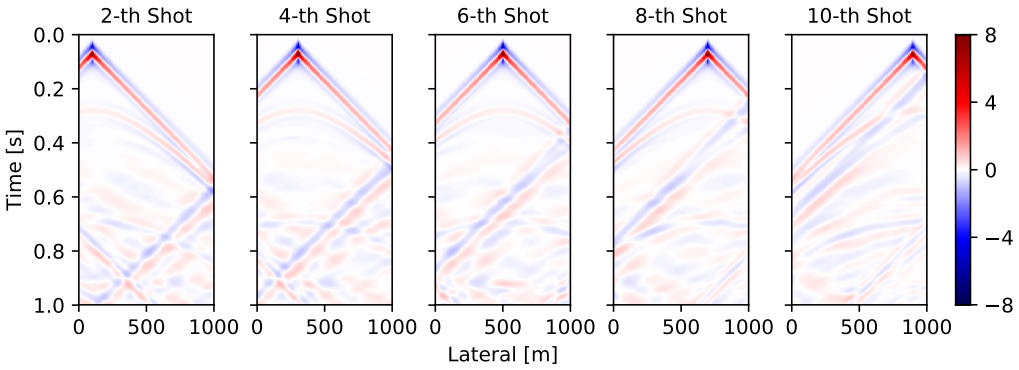

FIG. 4. The shot record calculated by the Born modelling method.

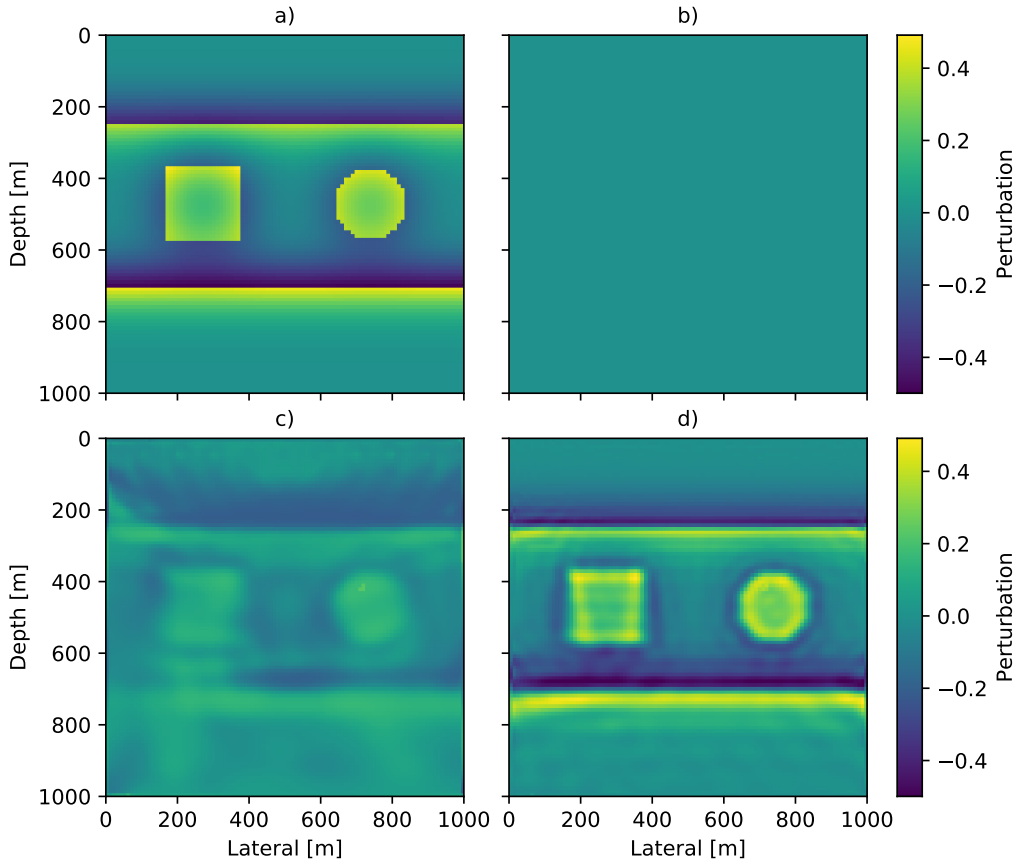

FIG. 5. The updated model at specific iterations. a) The true model; b) The initial zero model; c) The estimated model at the 10th iteration with ADAM optimizer using a learning rate of 0.3; d) The model at the 50th iteration.

The inversion results

Now we test whether the RNN is capable of calculating model corrections from the prediction errors, that is the RNN back propagation algorithm. The shot record shown in Figure 4 was feed to the RNN as the desired output (target). At each iteration, the training process automatically finds the inverse of the prediction error and returns an estimate of the model. Then the same test was done on the more structured Marmousi model (Figure 7). In our tests we found that the method does not seem to reach a stopping criteria. This is because the high frequency component of the model is difficult to recover, decreasing the convergence rate with iterations. After hundreds of iterations the convergence rate is so small that, after a reasonable period of time, the cost function does not reach the accuracy threshold. To circunvent this issue, we have to set a maximum for the number of iterations that unfortunately depends on the chosen optimization method. The following subsections show results of using the ADAM optimizer.

The scattering model

Figure 5 shows the inversion result by the ADAM optimizer applied to the scattering model shown in Figure 2. The initial model was all zeros and the learning rate was set to

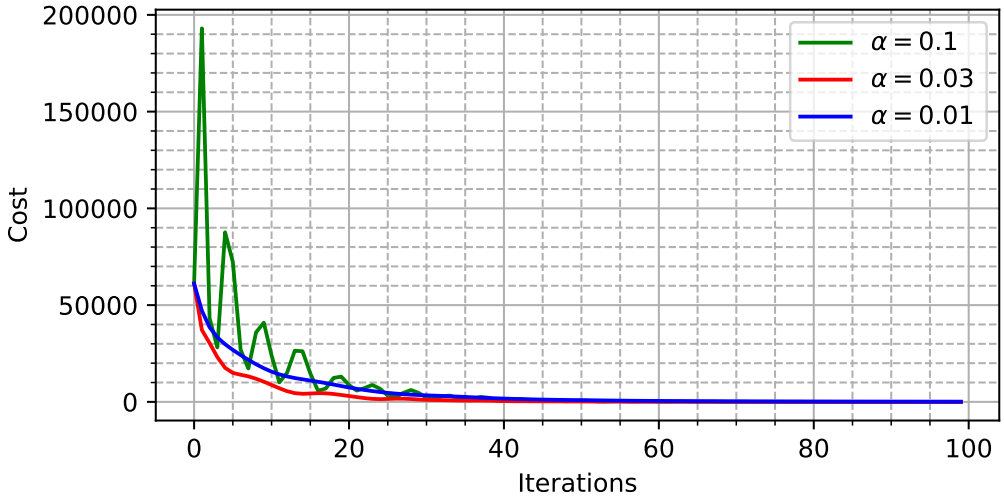

FIG. 6. The cost functions for different value of α with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ in all cases.

a value of 0.03. We can see that the main skeleton of the model was partially recovered after 10 iterations. The model kept improving and gained more high frequencies through iterations. Figure 6 shows the cost function response to different choices of the learning rate α. As mentioned in the theory section, β_1 and β_2 are weighing factor that determine how much the update is related to previous gradients, while the stepsize is controlled by α. Since the absolute value of the update is never greater than α, the choice of α is crucial and problem-specific. For the scattering model shown in Figure 2, velocity perturbations range from -0.4 to 0.4. If setting the number of iterations to 400 and letting the stepsize to be equal to α, then α should be around $0.4/400 = 0.01$. In Figure 6, we can see that $\alpha = 0.01$ is too conservative—the curve is smooth but it converges slower then $\alpha = 0.03$. In contrast, $\alpha = 0.01$ seems to be too aggressive—the oscillation on the curve indicates the stepsize used is too large. Although it still approaching the local minima, with satisfying speed, the progress is at high risk as the function may get trap into other local minima at any oscillation peaks. $\alpha = 0.03$, despite some minor imperfection, converges the fastest to the local minimum. Furthermore, the training process does not need 400 iterations to converge and it gives good results after 50 iterations. Selected iterations of model was shown in Figure 5b, 5c and 5d.

The Marmousi model

One may find that even a non-linear method like FWI has difficulty to estimate flat layers since the high-frequency part of the model takes long time to converge. Therefore, we test a different model, Marmousi (see Figure 7), which although more complex is more representative of the kind of problems we would like to solve. The model grid has a dimension of 94×288 with a cell size of $10\,\mathrm{m}$. For our tests, we reduced the depth and lateral distance by a constant scale factor to reduce the memory usage. Otherwise, the model would need a lower dominant frequency to satisfy the dispersion condition and would result in images with poor resolution. Figure 8 shows the model updates obtained at selected iterations using the ADAM optimizer with a learning rate of $\alpha = 0.03$. The red line in Figure 9 refers to the cost function corresponding to it. The values of the cost function are, at the initial

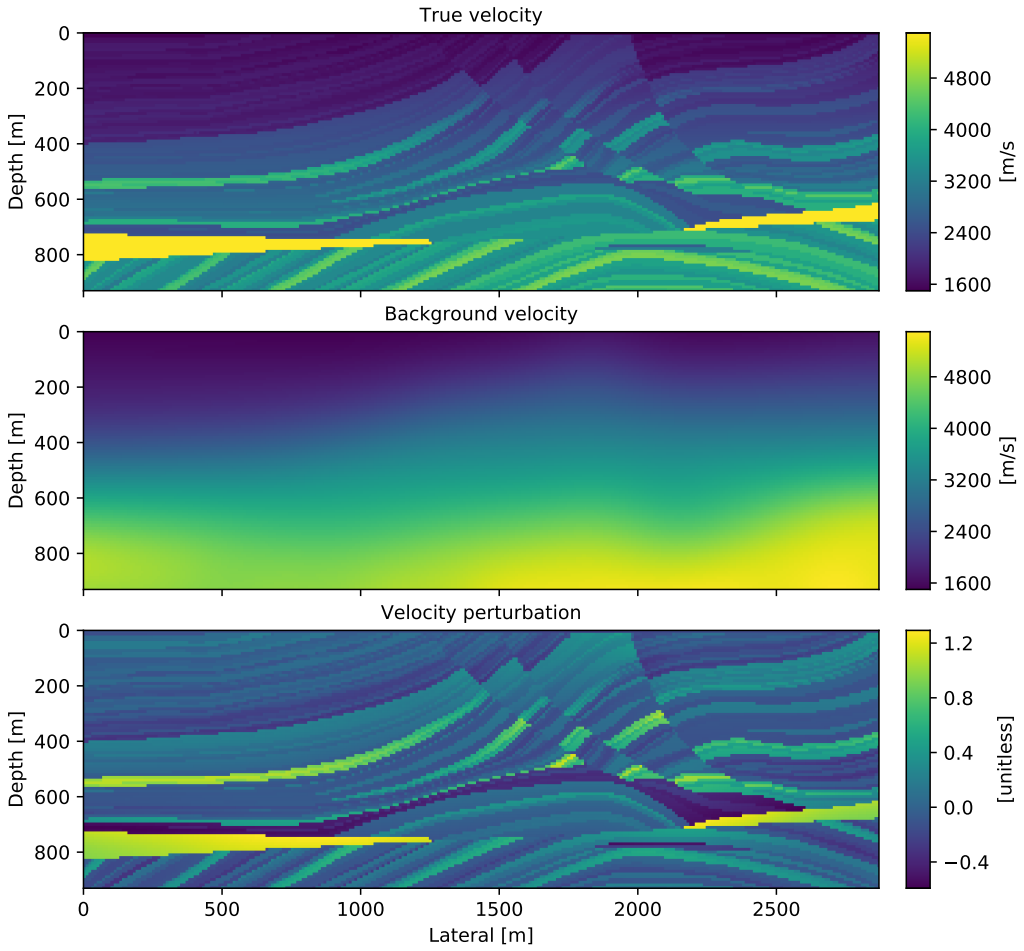

FIG. 7. The Marmousi model. The background velocity model is obtained by gaussian filtering of the true model. The velocity perturbation is $2\delta v/v_0$, as defined in the theory section.

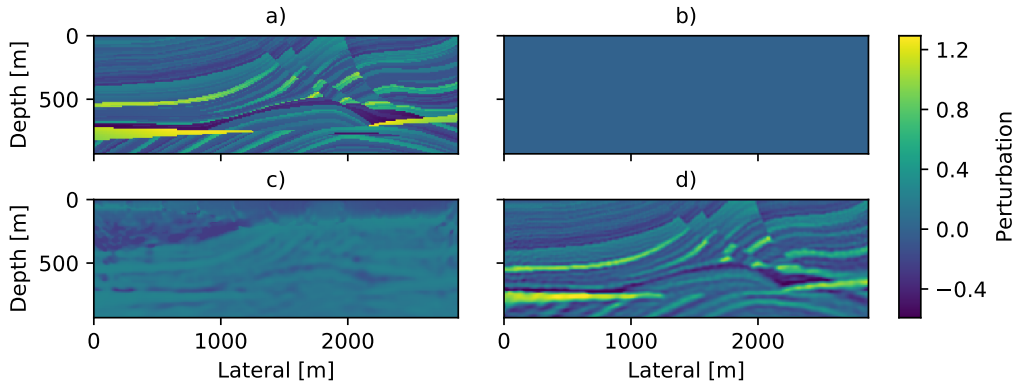

FIG. 8. The inversion results of Marmousi model by RNN. a) The true model; b) The initial zero model; c) The estimated model at the 10th iteration with ADAM optimizer using learning rate 0.3; d) The model at the 50th iteration.

state equal to 1933903.87, at the 200th iteration equal to 247.67, and at the 300th iteration equal to 101.88.

Non-linear optimizers

Two non-linear optimization methods were used in this report, the FR-CG and the L-BFGS-B method. Both of the methods use line search, which means the cost will be computed more than once at some points. The green and blue lines in Figure 9 show cost functions for both methods, respectively.

For the FR-CG method, it is noticeable that some severe line search oscillation happens in early iterations and cause it to be slow from the beginning. The cost is 33246.41 at the 200th function evaluation and converges extremely slow after the cost is minimized to around 2000.

For the L-BFGS-B method, there are only minor line search oscillations. The method is a lot faster than the FR-CG method and converges to a lower cost. The cost is 4044.99 at the 200th function evaluation and converges to around 300 eventually. However, neither of the methods outperforms the ADAM optimizer. This is because the step length is problem-specific and gives a boost to the optimization process.

Limitations

Although RNN is convenient to take advantage of parallel CPUs and GPU acceleration, it also has many drawbacks. On the one hand, RNN uses small number of time steps (usually less than a 100), because of the kind of problems it was designed for (voice recognition and word processing). On the other hand, it needs to use small time steps to avoid aliasing and distortion and to fully cover realistic models. TensorFlow saves all the functions to cache for each layer before starting the back-propagation, but in the kind of problems discussed in this report there are too many variables. For the scatter model as shown in Figure 2, though the actual training progress is fast, it takes $10\,\mathrm{gigabytes}$ of memory and usually needs $15\,\mathrm{min}$ to $20\,\mathrm{min}$ just for the initial setup. This disadvantage may be avoided

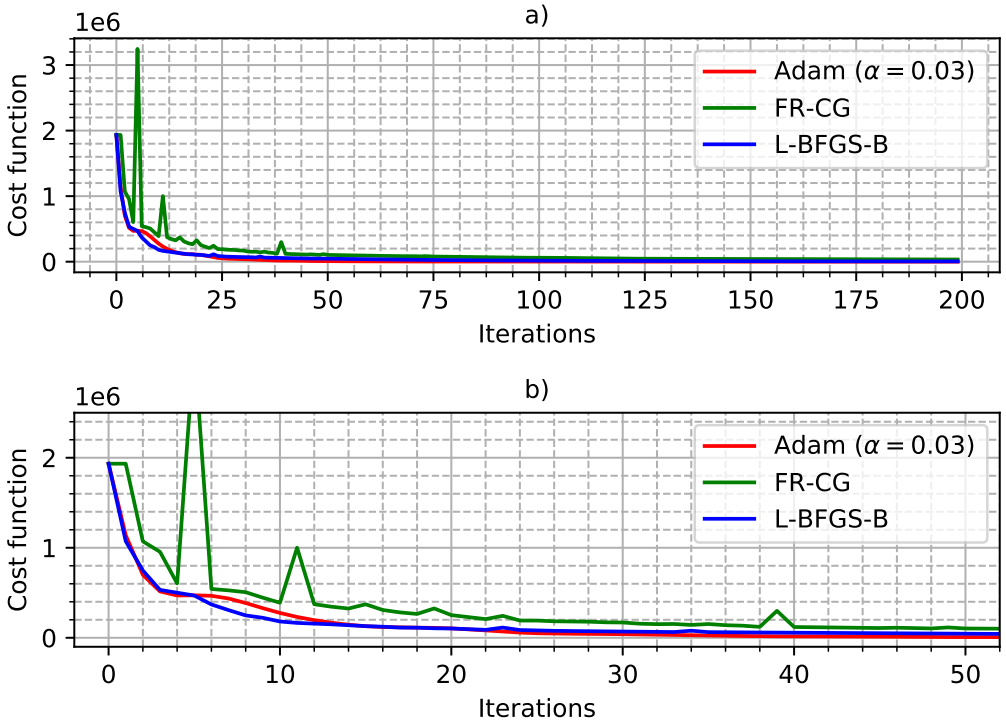

FIG. 9. Cost function curves comparison of the three used methods. a) The first 200 times of loss calculations; b) a zoomed version of the figure a.

by designing a more suitable neural network structure or by transforming the time domain into the frequency domain to reduce the number of layers in RNN.

The other notable drawback is from the use of the Born approximation. Like LSRTM, that also uses the Born approximation, the method also highly depends on the quality of the background velocity estimation. A poor/wrong background velocity will result in position shifts for the imaged reflector. A solution for this would be adding more non-linearity. For example, the forward modelling can be replaced by the finite difference method, or we can update the background velocity during the iteration to add more non-linearity.

CONCLUSION

The Born modelling can be successfully implemented using RNN with TensorFlow. Then, by feeding a theoretical data to the RNN built, the model can be inverted by back-propagation of the RNN. This operation can be proven to be the same as the LSRTM formulation. We found the ADAM method seems to be the most efficient optimizer but it requires to be extra careful when choosing the hyper-parameters. The second efficient optimizer is L-BFGS-B, which does not take extra hyper-parameters. The least efficient optimizer in our tests is the FR-CG, which spends much time in line searching and hence causes too many perturbations to the loss curve. The overall computing performance is good but TensorFlow takes too much time and memory to build the network before the back-propagation. In the future, we are interested in bringing this method to the frequency domain and looking for a more suitable neural network structure for wave propagation.

ACKNOWLEDGEMENTS

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: THE DERIVATION OF THE GRADIENT

First, let us express the current perturbation wavefield with the two previous perturbation wavefields at time step t, $t+1$ and $t+2$, respectively. We get

$$\delta \boldsymbol{p}^{(t)} = \left(2 + \Delta t^2 {\boldsymbol{v}_0}^2 \nabla^2\right) \delta \boldsymbol{p}^{(t-1)} - \delta \boldsymbol{p}^{(t-2)} + \Delta t^2 \cdot \boldsymbol{m} \cdot \frac{\partial^2 \boldsymbol{p}_0^{(t-1)}}{\partial t^2} \tag{A.1a}$$

$$\delta \boldsymbol{p}^{(t+1)} = \left(2 + \Delta t^2 {\boldsymbol{v}_0}^2 \nabla^2\right) \delta \boldsymbol{p}^{(t)} - \delta \boldsymbol{p}^{(t-1)} + \Delta t^2 \cdot \boldsymbol{m} \cdot \frac{\partial^2 \boldsymbol{p}_0^{(t)}}{\partial t^2} \tag{A.1b}$$

$$\delta \boldsymbol{p}^{(t+2)} = \left(2 + \Delta t^2 {\boldsymbol{v}_0}^2 \nabla^2\right) \delta \boldsymbol{p}^{(t+1)} - \delta \boldsymbol{p}^{(t)} + \Delta t^2 \cdot \boldsymbol{m} \cdot \frac{\partial^2 \boldsymbol{p}_0^{(t+1)}}{\partial t^2} \tag{A.1c}$$

where $\delta \boldsymbol{p}$ refers to the perturbation wavefield and the superscript refers to the corresponding time step. $\boldsymbol{p}_0$ is the background wavefield. $\boldsymbol{m}$ refers to the velocity perturbation ($2\delta \boldsymbol{v}/\boldsymbol{v}_0$). Note that the last term in (A.1a) is the source term scaled by $\Delta t^2 {\boldsymbol{v}_0}^2$. This characteristic is used later in the proof.

Differentiate (A.1a) with respect to $\boldsymbol{m}$ and differentiate (A.1b) and (A.1c) with respect to $\delta \boldsymbol{p}^{(t)}$. Then we get

$$\frac{\partial \delta \boldsymbol{p}^{(t)}}{\partial \boldsymbol{m}} = \Delta t^2 \frac{\partial^2 \boldsymbol{p}_0^{(t-1)}}{\partial t^2} \tag{A.2a}$$

$$\frac{\partial \delta \boldsymbol{p}^{(t+1)}}{\partial \delta \boldsymbol{p}^{(t)}} = 2 + \Delta t^2 {\boldsymbol{v}_0}^2 \nabla^2 \tag{A.2b}$$

$$\frac{\partial \delta \boldsymbol{p}^{(t+2)}}{\partial \delta \boldsymbol{p}^{(t)}} = -1 \tag{A.2c}$$

From Equation 7, the cost of each shot at a specific time slice is

$$J^{(t)} = \frac{1}{2}\left(\boldsymbol{D}^{(t)} - {\boldsymbol{d}_{cal}}^{(t)}\right)^2 = \frac{1}{2}\left({\boldsymbol{d}_{obs}}^{(t)} - \mathcal{S}_{\boldsymbol{x}_r} \delta \boldsymbol{p}^{(t)}\right)^2 \tag{A.3}$$

where $\mathcal{S}_{\boldsymbol{x}_r}$ is the sampling operator that extract the data from the wavefield at the receiver positions to form the shot record $\boldsymbol{d}_{cal}$. By taking the derivative with respect to the current perturbation wavefield $\delta \boldsymbol{p}^{(t)}$ on both side, we can get

$$\frac{\partial J^{(t)}}{\partial \delta \boldsymbol{p}^{(t)}} = -\mathcal{S}_{\boldsymbol{x}_r}\left(\boldsymbol{D}^{(t)} - \mathcal{S}_{\boldsymbol{x}_r} \delta \boldsymbol{p}^{(t)}\right) \tag{A.4a}$$

$$= -\mathcal{S}_{\boldsymbol{x}_r} \boldsymbol{r}^{(t)} \tag{A.4b}$$

$$= -\boldsymbol{r}^{(t)} \tag{A.4c}$$

Since $\mathcal{S}_{\boldsymbol{x}_r}$ is the sampling operator, $\mathcal{S}_{\boldsymbol{x}_r} \boldsymbol{r}^{(t)}$ will simply be $\boldsymbol{r}^{(t)}$. Similar to what is discussed by Richardson (2018). The gradient of the const function with respect to tone

wave field at specific time step t can be express as

$$\begin{aligned}\left.\frac{\partial J}{\partial \delta \boldsymbol{p}}\right|_t = & \left.\frac{\partial J}{\partial \delta \boldsymbol{p}}\right|_{t+2} \cdot \frac{\partial \delta \boldsymbol{p}^{(t+2)}}{\partial \delta \boldsymbol{p}^{(t)}} \\ & + \left.\frac{\partial J}{\partial \delta \boldsymbol{p}}\right|_{t+1} \cdot \frac{\partial \delta \boldsymbol{p}^{(t+1)}}{\partial \delta \boldsymbol{p}^{(t)}} \\ & + \frac{\partial J^{(t)}}{\partial \delta \boldsymbol{p}^{(t)}}\end{aligned} \tag{A.5}$$

According to the chain rule, the gradient of the cost J with respect to $\boldsymbol{m}$ at a specific time step t can be express as

$$\left.\frac{\partial J}{\partial \boldsymbol{m}}\right|_t = \left.\frac{\partial J}{\partial \delta \boldsymbol{p}}\right|_t \cdot \frac{\partial \delta \boldsymbol{p}^{(t)}}{\partial \boldsymbol{m}} \tag{A.6}$$

By Substituting (A.5), (A.2b), (A.2c) and (A.2a) into (A.6), the equation becomes

$$\left.\frac{\partial J}{\partial \boldsymbol{m}}\right|_t = \left[\left(2 + \Delta t^2 {\boldsymbol{v}_0}^2 \nabla^2\right) \left.\frac{\partial J}{\partial \delta \boldsymbol{p}}\right|_{t+1} - \left.\frac{\partial J}{\partial \delta \boldsymbol{p}}\right|_{t+2} + \frac{\partial J}{\partial \delta \boldsymbol{p}^{(t)}}\right] \cdot \left[\Delta t^2 \frac{\partial^2 \boldsymbol{p}_0^{(t-1)}}{\partial t^2}\right] \tag{A.7}$$

where the last term in the first bracket equals to the residual time slice (Equation A.4c). If we compare the terms in the first bracket with terms in Equation A.1a, we can notice that $\left.\frac{\partial J}{\partial \delta \boldsymbol{p}}\right|_t$ is actually a wavefield at time step $t+1$ which uses the residual slice scaled by $1/\Delta t^2 {\boldsymbol{v}_0}^2$ as source, denoted by $\boldsymbol{B}(\boldsymbol{r}^{(t+1)}/\Delta t^2 {\boldsymbol{v}_0}^2, \boldsymbol{x}, t+1)$. $\boldsymbol{B}$ refers to a time-reverse-propagation of wavefield since the wavefield at time step t is calculated by the wavefields at future time steps $t+1$ and $t+2$. The term in the 2nd time derivative of the background wavefield ($\boldsymbol{p}_0$) scaled by Δt^2 at time step $t-1$, denoted by $\Delta t^2 \boldsymbol{F}(f, \boldsymbol{x}, t-1)$. $\boldsymbol{F}$ is the time-forward-propagation of wavefield since $\boldsymbol{p}_0$ is propagating in the positive time direction. Then, Equation A.7 becomes

$$\left.\frac{\partial J}{\partial \boldsymbol{m}}\right|_t = -\boldsymbol{B}(\boldsymbol{r}^{(t+1)}/\Delta t^2 {\boldsymbol{v}_0}^2, \boldsymbol{x}, t+1) \cdot \Delta t^2 \boldsymbol{F}(f, \boldsymbol{x}, t-1) \tag{A.8a}$$

$$\approx -\frac{1}{{\boldsymbol{v}_0}^2} \boldsymbol{B}(\boldsymbol{r}^{(t+1)}/\Delta t^2 {\boldsymbol{v}_0}^2, \boldsymbol{x}, t+1) \cdot \boldsymbol{F}(f, \boldsymbol{x}, t-1) \tag{A.8b}$$

With Equation A.8b, we can infer the gradient in the global scale instead of at individual time step

$$\left.\frac{\partial J}{\partial \boldsymbol{m}}\right|_t \approx -\frac{1}{{\boldsymbol{v}_0}^2} \boldsymbol{B}(\boldsymbol{r}, \boldsymbol{x}, T-t) \cdot \frac{\partial^2}{\partial t^2} \boldsymbol{p}_0(f, \boldsymbol{x}, t) \tag{A.9}$$

which is similar to the form of LSRTM.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X., 2015, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org.
URL `https://www.tensorflow.org/`

Fukushima, K., 1979, Neural network model for a mechanism of pattern recognition unaffected by shift in position-neocognitron: IEICE Technical Report, A, **62**, No. 10, 658–665.

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

Karpatne, A., Atluri, G., Faghmous, J. H., Steinbach, M., Banerjee, A., Ganguly, A., Shekhar, S., Samatova, N., and Kumar, V., 2017, Theory-guided data science: A new paradigm for scientific discovery from data: IEEE Transactions on Knowledge and Data Engineering, **29**, No. 10, 2318–2331.

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

LeCun, Y., Bengio, Y., and Hinton, G., 2015, Deep learning: nature, **521**, No. 7553, 436.

Lipton, Z. C., Berkowitz, J., and Elkan, C., 2015, A critical review of recurrent neural networks for sequence learning: arXiv preprint arXiv:1506.00019.

Moseley, B., Markham, A., and Nissen-Meyer, T., 2018, Fast approximate simulation of seismic waves with deep learning: arXiv preprint arXiv:1807.06873.

Richardson, A., 2018, Seismic full-waveform inversion using deep learning tools and techniques: arXiv preprint arXiv:1801.07232.

Wright, S., and Nocedal, J., 1999, Numerical optimization: Springer Science, **35**, No. 67-68, 7.