Inversion with the Born approximation in a deep learning framework

Zhan Niu, Jian Sun, Daniel O. Trad

CREWES/University of Calgary

Summary

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 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 the 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 a 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 geophysics (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

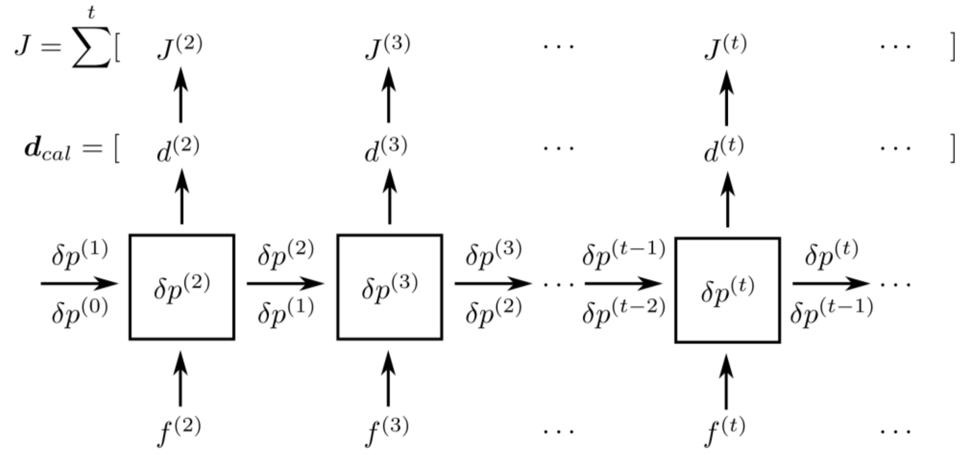
The Born modelling is governed by the following equations

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

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

where $\boldsymbol{p}_0(\boldsymbol{x},t)$ is the wavefield that propagate in a background velocity $\boldsymbol{v}_0(\boldsymbol{x})$ with a source $\boldsymbol{f}(\boldsymbol{x},t)$. $\boldsymbol{m}(\boldsymbol{x})$ refers to the model parameter which is defined as $\frac{2\delta\boldsymbol{v}}{\boldsymbol{v}_0}$. $\delta\boldsymbol{p}(\boldsymbol{x},t)$ refers to the perturbation wavefield that cause by the velocity perturbation $\boldsymbol{m}$. The sum of $\boldsymbol{p}_0$ and $\delta\boldsymbol{p}$ can be considered as an approximation of the total wavefield $\boldsymbol{p}(\boldsymbol{x},t)$ if $\boldsymbol{m}$ is assumed to be small.

Those two equations can then be implemented by RNN with the help of TensorFlow (Abadi et al., 2015). Figure 1 shows the RNN architecture used for this report.



$J=\sum^{t}[\quad J^{(2)}\quad J^{(3)}\quad\cdots\quad J^{(t)}\quad\cdots\quad]$
$\boldsymbol{d}_{cal}=[\quad d^{(2)}\quad d^{(3)}\quad\cdots\quad d^{(t)}\quad\cdots\quad]$
$\delta p^{(1)}$ $\delta p^{(0)}$
$\delta p^{(2)}$
$\delta p^{(2)}$ $\delta p^{(1)}$
$\delta p^{(3)}$
$\delta p^{(3)}$ $\delta p^{(2)}$
$\delta p^{(t-1)}$ $\delta p^{(t-2)}$
$\delta p^{(t)}$
$\delta p^{(t)}$ $\delta p^{(t-1)}$
$f^{(2)}$ $f^{(3)}$ $\cdots$ $f^{(t)}$ $\cdots$

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 the superscript (t) denote a variable at the t^{th} time step. 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, $\boldsymbol{d}^{(t)}$ can be extracted from the wavefield and the cost function $J^{(t)}$ can be calculated. Then the wavefields at (t) and $(t-1)$ will be forwarded 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 concatenating $\boldsymbol{d}^{(t)}$ from each cell and the cost function J can also be computed by $\mathrm{J} = \sum_{\mathrm{t}}(\boldsymbol{d}_{cal} - \boldsymbol{D})$ if a measured data $(\boldsymbol{D})$ is given. The gradient $(\partial J/\partial \boldsymbol{m})$ can be proven to have the same format as an RTM, by following similar derivation in Sun et al., 2019:

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

Results

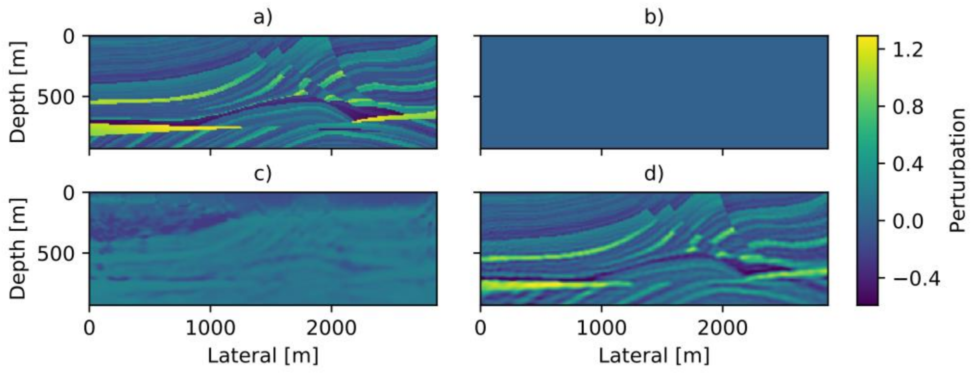


FIG. 2. The inversion results of Marmousi model by RNN. a) The true model; b) The initial zero model; c) The estimated model at the 10th iteration with ADAM optimizer using learning rate 0.3; d) The model at the 50th iteration.

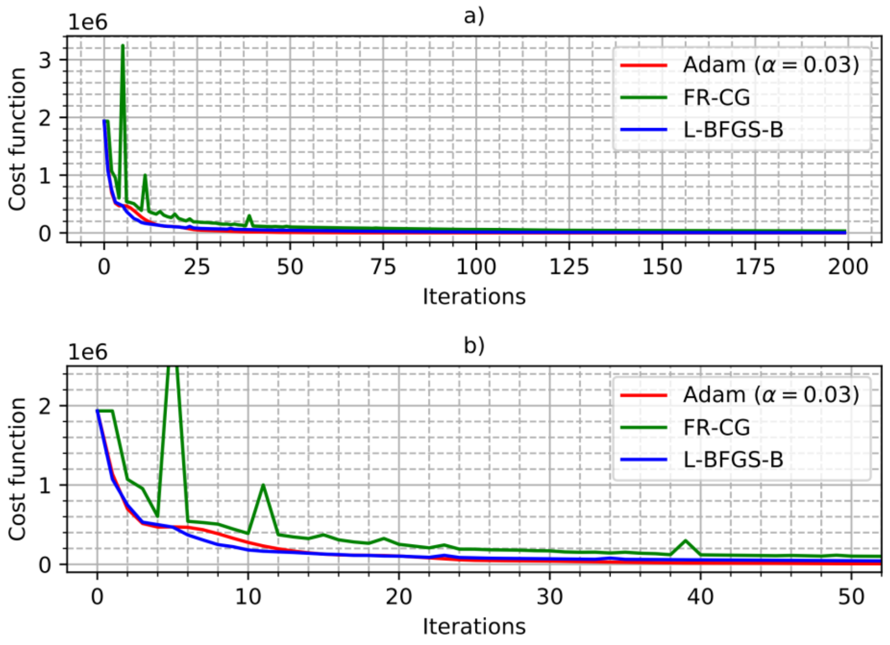


FIG. 3. Cost function curves comparison of the three used methods. a) The first 200 times of loss calculations; b) a zoomed version of the figure a.

The Marmousi model was used to test the Born inversion RNN. The model grid has a dimension of 94×288 with a cell size of 10m. 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 2 shows the model updates obtained at selected iterations using the ADAM optimizer (ADAptive Moment, Kingma and Ba, 2014) with a learning rate of $\alpha = 0.03$. The red line in Figure 3 refers to the cost function corresponding to it. The values of the cost function are, at the initial state equal to 1933903.87, at the 200th iteration equal to 247.67, and at the 300th iteration equal to 101.88. Additionally to Adam method, two non-linear optimization methods are compared and summarized in Figure 3. The first method is Fletcher-Reeves conjugate gradient (FR-CG, Wright and Nocedal, 1999). 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. The second non-linear method in Figure 3 is Broyden–Fletcher–Goldfarb–Shanno method (L-BFGS-B, Wright and Nocedal, 1999). 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.

Although the RNN implementation in TensorFlow was a convenient testing platform for this initial test, allowing us to take advantage of parallel CPUs and GPU acceleration, it also has many drawbacks. On the one hand, RNNs usually use a 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 a typical horizontal layered model, though the actual training progress is fast, it takes 10 gigabytes of memory and usually needs 15 min to 20 min just for the initial setup. This disadvantage may be avoided 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.

Conclusions

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 backpropagation 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.

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.

Sun, J., Niu, Z., Innanen, K., Li, J., Trad, T., 2019, A deep learning perspective of the forward and inverse problems in exploration geophysics: CSEG/CSPG/CWLS/GeoConvention 2019.

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