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

Bingbing Sun and Tariq Alkhalifah, King Abdullah University of Science and Technology

SUMMARY

Full-Waveform Inversion is a nonlinear inversion problem, and a typical optimization algorithm such as nonlinear conjugate-gradient or LBFGS would iteratively update the model along gradient-descent direction of the misfit function or a slight modification of it. Rather than using a hand-designed optimization algorithm, we trained a machine to learn an optimization algorithm which we refer to as "ML-descent" and applied it in FWI. Using recurrent neural network (RNN), we use the gradient of the misfit function as input for training and the hidden states in the RNN uses the history information of the gradient similar to an BFGS algorithm. However, unlike the fixed BFGS algorithm, the ML version evolves as the gradient directs it to evolve.The loss function for training is formulated by summarization of the FWI misfit function by the L2-norm of the data residual. Any well-defined nonlinear inverse problem can be locally approximated by a linear convex problem, and thus, in order to accelerate the training speed, we train the neural network using the solution of randomly generated quadratic functions instead of the time-consuming FWI gradient. We use the Marmousi example to demonstrate that the ML-descent method outperform the steepest descent method, and the energy in the deeper part of the model can be compensable well by the ML-descent when the pseudo-inverse of the Hessian is not incorporated in the gradient of FWI.

INTRODUCTION

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

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

There are various iterative methods for solving the nonlinear optimization problem of FWI including the steepest descent method, nonlinear conjugate-gradient method, and Newton's methods, like the Gauss-Newton method , and various quasi-Newton methods (Ma and Dave, 2012). Compared to the steepest descent and nonlinear conjugate-gradient method, the Newton's methods generally converge faster in fewer iterations. However, for a model size $\mathscr{O}(N)$, the full Newton's methods require evaluating the inverse of the Hessian matrix of size in $\mathscr{O}(N^2)$, which is computationally prohibitive especially for 3D problems. Thus, various methodologies have been proposed to approximate the Hessian matrix. For example, the Gaussian-Newton method ignores second-order terms that account for nonlinearity in the misfit function. Quasi-Newton methods do not compute the Hessian matrix, but instead iteratively update a Hessian approximation. Different strategies for such Hessian updating can be utilized, such as the BroydenFletcherGoldfarbShanno (BFGS) algorithm (Fletcher, 1987) and the DavidonFletcherPowell (DFP) algorithm (Davidon, 1991). However, although the BFGS method reduces the computation time required to approximate a Hessian matrix, it does not decrease the amount of memory required to store the approximated Hessian.

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

All the aforementioned hand-crafted optimization algorithms are applicable to general inverse problems, and is not specific to FWI. In this abstract, we propose to design an optimization algorithm for FWI based on Machine Learning (ML). Specially, we take the gradient of the misfit function at the current step as input and utilize the recurrent neural network (RNN) for training. RNN has wide applications in machine learning such as natural language understanding, language generation, video processing and may other tasks. Recently, Andrychowicz et al. (2016) proposed to use RNN for designing optimization algorithms. Their learned algorithms outperform generic, hard-designed alternatives on the tasks for which they are trained, and also generalize well to new tasks with a similar structure. Thus, the majority of the work presented in this abstract is inspired by their work. The difference is that they applied the learned algorithms to optimize the machine learning problem such as for classification, while we applied the learned algorithm to a specific inverse problem in geoscience, i.e, the full-waveform inversion here. Our main purpose is to figure out whether the machine can learn how to better update the model for an optimization problem, and thus help in accelerating the convergence of FWI. We test the approach on the Mamrousi model, and share the results.

MACHINE LEARNING OF AN OPTIMIZATION ALGORITHM

A first-order optimization algorithm such as nonlinear conjugate-gradient or LBFGS method, can be summarized as

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \gamma \mathbf{d}_{t-1}(\mathbf{g}, \mathbf{d}), \tag{1}$$

where γ is the learning rate (step length), x_t is the variable we want to invert for in the original inversion problem. The vector $\mathbf{d}_{t-1}$ is the updating direction, which is a function of the gradient $\mathbf{g}$ and the updating direction $\mathbf{d}$ at the previous and current steps. The difference between the different optimization algorithms, e.g., nonlinear conjugate-gradient or LBFGS is how to compute the updating direction based on the history of the updating information.

Now, rather than using already existing algorithms, we try to find a function using neural network, which can map the current gradient $\mathbf{g}$ into the updating direction $\mathbf{d}$. In order to utilize the history of the updating information, i.e., the previous gradients $\mathbf{g}$ and updating directions $\mathbf{d}$ (so the resulting optimization algorithm becomes similar to the LBFGS method), we use a special network called recurrent neural network (RNN). As shown in Figure 1, its input is the gradient $\mathbf{g}_t$ and the hidden state $\mathbf{h}_t$ at current time step, and its output would be the updating direction $\mathbf{d}$ and hidden state $\mathbf{h}_{t+1}$ for next time iteration:

$$(\mathbf{d}_t, \mathbf{h}_{t+1}) = \mathrm{RNN}(\mathbf{g}_t, \mathbf{h}_t; \phi), \tag{2}$$

where ϕ is the parameters of RNN hat we need to evaluate and optimize. From Figure 1, we can see that RNN is used recursively, note that we have only one neural network of RNN and in each time step, they share the same parameter ϕ for training, i.e., we only have one set of parameter ϕ during the training. We unroll along the time direction for demonstration purposes. The hidden state variable $\mathbf{h}_t$, which appears in both of the input and output of RNN, keeps track of the history of the updating information, and thus allows the resulting algorithm to use the history of the updating like in the LBFGS method and in fact many recent learning procedures in ML, such as adaptive moment estimation, also utilizes the history information by taking the momentum into consideration.

The loss function for training would be a weighted summation of the mean square errors of the residuals at all time steps:

$$L = \sum_t w_t f(x_t(\phi)), \tag{3}$$

where f is the misfit function for the original inverse problem. Since the updating direction is determined by the output of RNN, so f is a function of the parameter ϕ of the RNN. Note here, we have a weighted summation of the misfit function over all time steps (iterations). We can however, only select the final step of the misfit value as the loss function, but this would make the training to be difficult, because when back-propagating the residual, the gradient value reduces considerably when it propagates back to earlier time steps. Thus, in the examples shown in this abstract, we select the weighting w_t to be 1, i.e., we give the same weight for the misfit value in each step.

THE COORDINATE-WISE IMPLEMENTATION

For a typical 3D FWI application, the parameters of the model can be in the scale of millions. Optimizing at this scale with a fully connected RNN is not feasible as it would require huge hidden states and enormous number of parameters (note the parameters of the RNN would be in the order of $\mathcal{O}(N^2)$, where N is the size of the input). To resolve this difficulty, we use a modified RNN, which operates coordinate-wise on the parameters of the objective function as shown in Figure 2. Different coordinates share the same RNN parameters for updating while unique behavior on each coordinate is achieved by using separate hidden stable variables. Thus, this coordinate-wise network architecture can reduce the memory computation burden significantly without loss of the efficiency and accuracy of the resulting optimization algorithm.

TRAINING OF THE RNN

We can run FWI on different velocity models for training the parameters for RNN. However, a typical time-domain FWI requires solving a partial differential equation to construct the gradient and this tends to be computationally expensive. Thus, all the inversion problems are often approximated locally by a linear problem, we design various such linear problems for the training. One simple option, we define a quadratic misfit function f as:

$$f = ||\mathbf{W}\mathbf{x} - \mathbf{b}||_2^2. \tag{4}$$

We can randomly choose the matrix $\mathbf{W}$ and vector $\mathbf{b}$ to formulate a particular inverse problem. Such training is much more efficient as it avoids wave-equation modeling and alternatively here we only need one matrix multiplication and one vector subtraction for the modeling. For the examples shown here, we create 20,000 such linear inverse problem for training. The weighting matrix is chosen to be 10 by 10 and vector $\mathbf{b}$ is accordingly a 10-dimensional vector. Their elements are drawn randomly from a Gaussian distribution. Each function was optimized for 100 steps and every 20 steps we evaluated the summation of the misfit as defined in Equation 3 and back-propagated the residual for updating the parameters of the RNN. In Figure 3, we show the performance of different optimization algorithms on the randomly sampled 10-dimensional quadratic functions. We can see that the learned optimization algorithms by RNN show faster convergence than other state of the art algorithms such as ADAM (Kingma and Ba, 2014) and RMSprop(Ruder, 2016). This simple example clearly demonstrates the advantage of the learned optimization algorithms over hand-crafted ones. In the next section, we will apply this learned algorithm to the inversion of the modified Marmousi model.

EXAMPLES

In this section, we apply our learned optimization algorithm i.e., the ML-descent algorithm to invert for a modified Marmousi model. The true velocity v_{true} shown in Figure 4a extends 2 km in depth and 8 km, laterally. The initial velocity is shown in Figure 4b. The dataset is modeled using 100 shots with a source interval of 80 m and 200 receivers with an interval of 40 m. The source wavelet is a Ricker wavelet with a 6 Hz peak frequency. We perform a full-band dataset inversion and iterate for 20 iterations. In order to evaluate the performance of the optimization algorithm, we do not include the second order information e.g., the pseudo Hessian, in the gradient computation. Figure 4c to 4f shows the inverted model using the conventional steepest descent, Adam method, RMSprop method and the learned ML-descent methods, respectively. From the result, it is clear that without the pseudo Hessian to compensate for the illumination and geometric spreading effects, the steepest descent method can hardly recover the deeper part of the model. Adam method and RMSprop method result in improved result while the ML-descent method which is learned by a machine can successfully invert for the model both in the shallow and deeper parts and provides with the best result. The result of the ML descent includes some noise , which is probably due to the coordinstewise implementation of the RNN, which ignores the neighborhood information to make the resulting update smooth. This suggests that a modification of the current neural network architecture to change from a coordinate-wise implementation to patch-wise implementation, which can incorporate the neighborhood information, could be useful. At last, in Figure 5, we show the curves of the normalized misfit value over iterations, the good performance of the proposed ML-descent method is further demonstrated by its fast convergence.

CONCLUSION

We developped an optimization algorithm learned by a machine. RNN is used to learn an optimization algorithm, which can use the history of the update, like LBFGS. A coordinatewise implementation of the RNN is evaluated for efficient training of the network. Both the examples of a quadratic misfit function and the Marmousi model demonstrate the good performance of the proposed method. Improvements of current work can be made by considering a patch-wise implementation of the RNN or training the network for inversion of real FWI problems, both of which will be shared in the presentation of the work.

ACKNOWLEDGMENTS

We thank KAUST for the funding of this research and the members of SWAG group for useful discussions.

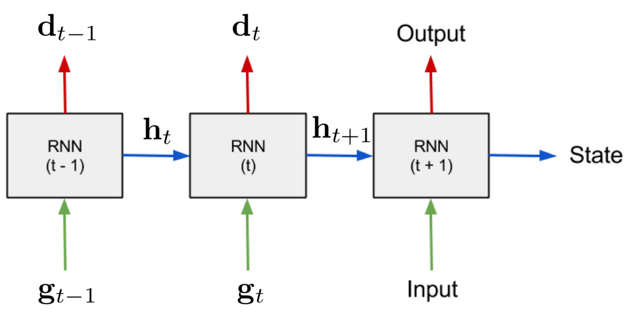

Figure 1: Demonstration of the RNN network: Input is the gradient **g** and output is the updating direction **d**, the hidden state **h** is used to keep track of and utilize the history information, i.e., $(\mathbf{d}_t, \mathbf{h}_{t+1}) = \mathrm{RNN}(\mathbf{g}_t, \mathbf{h}_t; \phi)$ where ϕ is the parameters for RNN which will be updated during training.

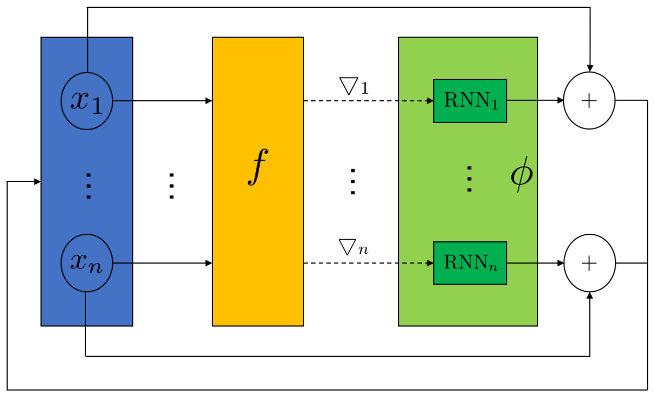

Figure 2: One step of coordinatewise RNN updating. All RNNs have the shared parameters ϕ, but separate hidden states.

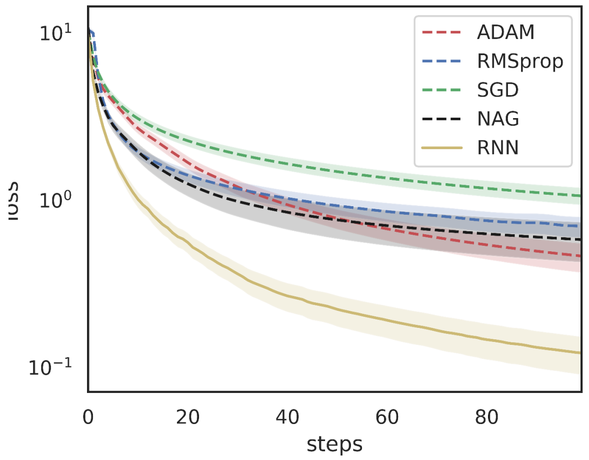

Figure 3: Performance of different optimization algorithms on randomly sampled 10-dimensional quadratic functions.The learned optimization algorithm by RNN shows faster convergence than other hand-crafted algorithms.

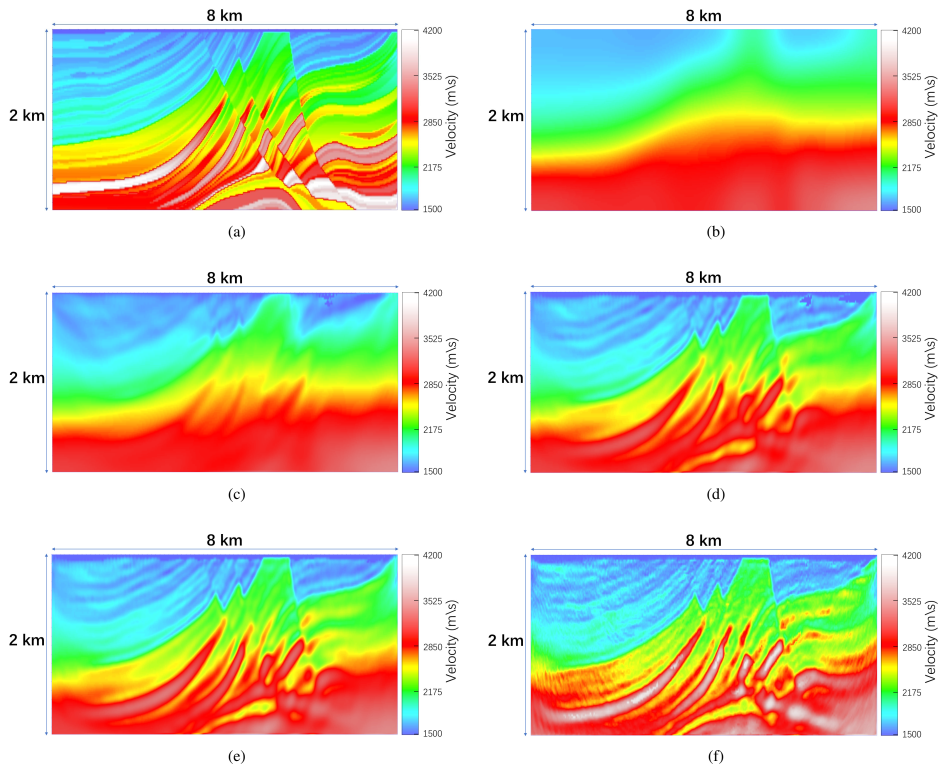

Figure 4: Marmousi velocity model for a) the true velocity; b) the initial velocity; the inverted velocity by c) the steepest descent method; d) Adam method; e) RMSprop method; f) the ML-descent method.

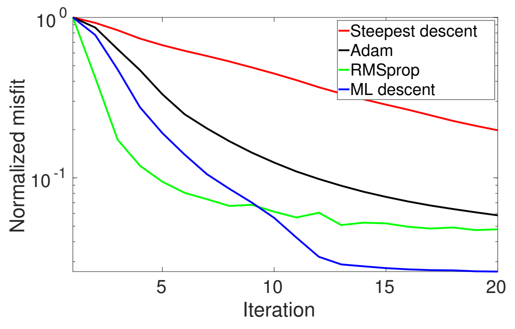

Figure 5: The curves of the normalized misfit over iterations: the learned ML-descent optimization algorithm shows faster convergence than the hand-drafted steepest descent algorithm and Adam method. Compared to RMSprop method, the ML-descent converges to a model with smaller data residual value.