Deep-LSRTM: least-squares reverse time migration via learned projection operators

Introduction

Least-squares migration (LSM) (Lailly and Bednar, 1983) seeks to overcome the limitations of adjoint-based imaging formulations by posing seismic migration as a linear inversion problem. Under the assumption of a sufficiently correct background velocity model, LSM can reduce migration artifacts, remove acquisition marks, compensate for illumination, and increase the resolution of seismic sections. Least-squares reverse time migration (LSRTM) (Wang et al., 2017) uses the RTM engine to perform demigration/migration sequences, delivering true amplitude images and sharpened subsurface reflectors. Conventional LSRTM implementations use iterative methods to obtain a reflectivity model that minimizes the $l_2$-norm misfit, defined in either the data or the image domain. Due to the ill-posed nature of the LSM problem, a regularization term is frequently incorporated to the misfit to impose prior information on the model (such as structure or sparseness constraints) and stabilize the inverted solution. Regularized LSRTM methods produce excellent results but can be challenging to deploy in practice due to the difficulty of hyper-parameter selection.

With the recent advent of deep learning applications, machine learning approaches have emerged as a powerful alternative for solving inverse problems. Most of the research on deep learning in seismic inversion leverages Deep Neural Networks (DNNs) to predict a subsurface model from seismic data in an end-to-end fashion, circumventing the need for iterative reconstruction (Sun et al., 2020; Araya-Polo et al., 2019; Li et al., 2020). However, with limited data, fully learned reconstructions are hard to train to achieve high-quality results and are inefficient for large image sizes due to memory limitations or the associated computational cost during training. Such restrictions stem from the fact that the network needs to learn a data-to-model space mapping physically described by non-trivial wave operators. We explore an alternative to circumvent these drawbacks by considering a learned iterative imaging algorithm, which explicitly incorporates a handcrafted forward-adjoint operator pair.

In this article, we formulate the pre-stack LSRTM problem using a deep-learned projected gradient descent scheme, which we have baptized as Deep-LSRTM. This scheme can be regarded as a DNN extension of the conventional projected gradient descent method. It substitutes the projection operator with sets of Convolutional Neural Networks (CNNs), learning an update function for each gradient descent iteration. The efficiency of the method is twofold: once trained, the networks can predict accurate reflectivity models with only a few iterations, and secondly, it implicitly learns the step size and the effect of regularization from the training data. In the following sections, we will first describe how the theory of LSRTM connects with Deep-LSRTM. Then we will present the proposed network architecture and the training procedure. After training, we evaluate our network’s performance on two synthetic examples. Finally, we present the conclusions and directions for future work.

Theory

The regularized LSRTM problem entails finding the vectorized reflectivity model $m$ that minimizes the following objective function:

$$E(m) = J(m) + \lambda R(m)$$

$$= \frac{1}{2} \| L m - d \|^2 + \lambda R(m),$$

(1)

where $L$ denotes the forward modeling (or demigration) operator that encapsulates the first order Born approximation of the two-way wave equation, $d$ represents the pre-stack recorded data, $R(m)$ is any generic regularization term, and $\lambda > 0$ is the damping parameter that controls the trade-off between the data fit $J(m)$ and $R(m)$. When $R(m)$ is differentiable, we can solve equation 1 iteratively with a simple gradient descent scheme

$$m_{k+1} = m_k - \alpha \left( \nabla J(m_k) + \lambda \nabla R(m_k) \right),$$

(2)

where

$$\nabla J(m_k) = L^T (Lm_k - d)$$

(3)
denotes the gradient of the data misfit, $\alpha$ is the step length, and $\nabla R(m_k)$ represents the gradient of the regularization. Conversely, if $R(m)$ is not differentiable or the constraints are hard to compute, we may reformulate the problem via a projected gradient descent algorithm by introducing a projection operator $\mathcal{P}$ that maps the model updates into a desired constrained space $\mathcal{C} : m = \mathcal{P}(m)$. The projected gradient descent method calculates model updates as

$$m_{k+1} = \mathcal{P}(m_k - \alpha \nabla J(m_k)).$$  \hspace{1cm} (4)

This structure inspires to replace any general projection that improves the current model update by a set of CNNs. Specifically, we can obtain optimal reflectivity updates by training the networks to make efficient projections following

$$m_{k+1} = \Lambda_{\theta_k}(m_k, \nabla J(m_k)),$$  \hspace{1cm} (5)

where each network $\Lambda_{\theta_k}$ represents a learned updating operator composed by $N$ stacked 2D convolutional layers and non-linear activation functions, as shown in Figure 1. As can be inferred from equation 5, the step length and the model features are implicitly learned from the data set during training instead of being explicitly computed. This is a sophisticated factor compared to traditional LSRTM, where it is often complicated to choose optimal projection or regularization strategies capable of expressing desirable geological and structural elements.

![Figure 1](image_url)

**Figure 1** The proposed network architecture for the learned projection operators $\Lambda_{\theta_k}$, based on an encoder-decoder scheme. The number of channels is shown below each convolutional layer. ReLU refers to the rectified linear unit function. Batch normalization (BN) is used to equalize the contributions from different inputs and avoid the vanishing gradient problem. The last convolutional layer does not use a ReLU function since the reflectivity coefficients can be positive or negative. We also add a skip connection between the input $m_k$ and the output layer, forcing the network to learn residual updates to the current iterate.

In our framework, each $\Lambda_{\theta_k}$ has the same architecture as the other networks, but each network possess its own set of learned parameters. The learning comes from a supervised training step that minimizes the mean squared error (MSE) loss between a set of ground-truth reflectivity images $m^3_{\text{true}}$ and the predicted outputs $m^3_{k+1}$, expressed as

$$\min_{\theta_k} \frac{1}{3} \sum_{i=1}^{3} ||m^i_{k+1} - m^i_{\text{true}}||^2_2,$$  \hspace{1cm} (6)

where $\mathfrak{S}$ indicates the total number of ground-truth reflectivity images in the training data set. Inspired by the work on Hauptmann et al. (2018), we train the projection operators $\Lambda_{\theta_k}$ sequentially in a greedy fashion, defining a fixed number of $K$ iterations. This means that the outputs predicted by a trained projection operator $\Lambda_{\theta_{k-1}}$ from a previous iteration will be part of the data set used to train $\Lambda_{\theta_k}$ in the next iteration. After training, the final reflectivity model is given by

$$m_K = (\Lambda_{\theta_{K-1}} \circ \Lambda_{\theta_{K-2}} \circ \ldots \circ \Lambda_{\theta_0})(m_0, \nabla J(m_0))$$

$$= \Lambda_{\Theta}(m_0, \nabla J(m_0)), \quad \text{with } \Theta = (\theta_0, \ldots, \theta_{K-1}).$$  \hspace{1cm} (7)

As previously stated, our model does not need to learn the forward or adjoint mappings. Instead, we explicitly compute the gradient $\nabla J(m_k)$ using equation 3 before feeding the current network with the inputs $(m_k, \nabla J(m_k))$, similar to classic optimization algorithms. This allows us to use shallower architectures, which are less prone to overfitting and require less training data.
Numerical Examples

To train our Deep-LSRTM framework, we first generate 1000 random velocity distributions of size 256x256 grid points with a regular grid spacing of 10 meters and then calculate the reflectivity as velocity perturbations in squared slowness units (Figure 2). Next, we separate 900 reflectivity models for training and 100 for validation. After hyper-parameter tuning, we set \( K = 5 \) and sequentially train each learned projection operator with 50000 iterations of Adam optimizer, using a learning rate of 0.001 and a batch of size 2. With these settings, the number of epochs per Deep-LSRTM iteration is 111. Figure 3 shows the training and validation loss functions per epoch, considering the 5 iterations of Deep-LSRTM. During training, the gradient calculation step is the most demanding part of our algorithm due to the demigration/migration sequence applied to each reflectivity model. Therefore, to reduce computational costs, we only simulate 15 shot gathers per model for imaging. The surface acquisition involves a fixed-spread geometry with 256 receivers evenly placed at 30 meters in depth and 10 meters spacing, and a 20 Hz Ricker wavelet used as the seismic source.

**Figure 2** Four examples of the reflectivity data set used as ground-truth for training. Each reflectivity is calculated from random synthetic velocity models ranging from 1500 to 5500 m/s. Velocity, folding amplitude, and fault size gradually increase with depth, and all models have a different number of layers. For simplicity, we impose a flat water bottom fixed at 100 meters.

**Figure 3** The normalized training and validation loss functions versus the number of epochs for 5 iterations of Deep-LSRTM. A similar pattern between both curves indicates that our method does not suffer from the overfitting problem.

Finally, we test our Deep-LSRTM approach on two synthetic examples. The first one is a model generated in the same way as the training samples but was not included in the training or validation sets. Figure 4.c show the result of the first Deep-LSRTM iteration. Our method’s deconvolution effect is evident because the wavelet signature is minimal compared to the RTM image (Figure 4.b). As can be noticed, the amplitude balance, structural continuity and resolution of the image greatly improve. Moreover, the projection operator satisfactorily learns to cancel reflectivity coefficients on the water layer. The first iteration result still preserves acquisition-related noise at the top layers and strong artifacts at the edges of the model. These are corrected with further iterations, as shown in Figure 4.d.

The second example comprises a more complex setting extracted from the central part of the Marmousi model (Martin et al., 2006). Similar to the first experiment, we display the RTM and Deep-LSRTM result in Figure 5. While we notice an overall improvement in the reflectivity models compared to the RTM result, the imaging quality is subpar at steep-dipping reflectors. This might be because structures with this kind of dipping angles are not contained in the training data set samples. For such cases, transfer learning could ameliorate the reconstruction quality for this part of the model at minimal extra cost.
Conclusions

We have built an LSRTM framework that leverages the universal approximation capabilities of CNNs to predict reflectivity updates by mimicking a projected gradient descent algorithm. Tests on synthetic data show that the iterative Deep-LSRTM approach yields high-quality results with accurate amplitudes and structure enhancement for regions of the model that share similar features as those learned during training. Future research will focus on implementing Deep-LSRTM in real seismic data and testing its performance considering wrong background velocity models and subsalt imaging.

Acknowledgements

The authors are grateful to the sponsors of the Signal Analysis and Imaging Group (SAIG) at the University of Alberta for their continued support.

References