Machine learning experiments on velocity extraction from migration images

Zhan Niu and Daniel O. Trad

ABSTRACT

In full waveform inversion (FWI), the update of velocity is obtained by calculating the gradient of the misfit between recorded and predicted data, which is defined by the cross-correlation of the reverse time of receiver wavefield and source wavefield. Benefits can be achieved by solving a direct non-linear mapping between the correlation and model update. In this report, we train a fully connected neural network with residual blocks which allows migrated images to be directly mapped into velocity models. The input images and the true velocity model comes from reverse time migration results on randomly generated 4-layer models. The training is performed with ADAM optimizer combined with L1/L2 norms as the loss function. Performance and convergence of the neural network with different hyper-parameters are also investigated systematically. We have tested the trained model with different synthetic inputs. Results show the that the trained network is relatively model dependent which performs well on the validation set but does a poor job on datasets that come from different distributions.

INTRODUCTION

Machine learning (ML) has become a great tool in the field of geophysics. It can overcome some drawbacks of the theory and could also bring the power of modern CPUs and GPUs because of its potential to parallelize. Machine learning has great power in numerical analysis, but it also has limitations. The most crucial limitation is about generality. Since machine learning is a data-driven technique, the trained network is often model-dependent and hard or invalid to be applied to other data. Especially in seismic inversion and forward modelling, there are tons of acquisition setups that can be adopted on a single velocity model, which brings great challenges to the training. Most of the researchers on solving modelling/inversion problems often assume a fixed acquisition geometry and train networks with data based on random velocity models. This generalizes the network to be working with velocity models in the real world, however, the process will be heavily dependent on the acquisition geometry.

In this report, we will explore a way to extract velocity from reverse time migration images. The reason why we use images from migration that it is best to try to remove the dependencies on the acquisition geometry. Shot gathers are a function of time, acquisition setups and the geology that lies beneath. Not all of them are of our interests — we care more about the structure and the rock properties. Migration, however, is a technique that tries to reduce the amount of less interesting parameters by mapping seismic events to reflector boundaries. Reverse time migration (Claerbout, 1971) uses the correlation of source wavefields and receiver wavefields to show the location of the reflector. This method requires a good background velocity model and does not deal with multiples (which is related to acquisition geometry), but has become very popular because of its accuracy. Different imaging conditions have been proposed and they all aim to achieve better accuracy for

estimating the reflectivity, from which we can extract more information about the subsurface. It is crucial to recover the correct reflectors' amplitude and it is beneficial for model updates in inversion methods such as FWI (Lailly and Bednar, 1983; Tarantola, 1984). Starting from the inaccurate amplitude from migration images, we trained a neural network to convert them into velocities at low costs.

THEORY

Reverse time migration

Reverse time migration (RTM) calculates the dot product of the up-going wavefield and down-going wavefield. Following Claerbout (1971), the reverse propagation of the shot record will coincide with the source wavefield at the position of the reflector. Most of the reverse time migrations follow this basic idea but there are many expressions of the correlation, which is called imaging condition. A typical 2D cross-correlation imaging condition can be defined as

$$I(\boldsymbol{x}) = \sum_t S(\boldsymbol{x}, t) \cdot R(\boldsymbol{x}, T - t) \tag{1a}$$

$$I'(\boldsymbol{x}) = \frac{2}{v_0(\boldsymbol{x})} \sum_t \frac{\partial^2 S(\boldsymbol{x}, t)}{\partial t^2} \cdot R(\boldsymbol{x}, T - t) \tag{1b}$$

where S refers to the source wavefield while R refers to the receiver wavefield. The dot in between refers to element-wise production. Then all images are superposed together to reduce the dependency on time. Equation 1a does not yield the true reflectivity but will be the easiest and most stable imaging condition to calculate. In this report, we define the imaging condition by Equation 1b as it is closer to the FWI gradient. The source wavefield is replaced by its 2nd derivative with respect to time and the amplitude is normalized by the background velocity model $v_o(\boldsymbol{x})$.

L1 norm and L2 norm

L1 norm and L2 norm (or its square) are the most common loss functions used in optimization problems. The choice between them can leads to different results in deconvolution problems (Taylor et al., 1979). Suppose the observation/true data is $\boldsymbol{y}$ and prediction is $\boldsymbol{y}_{\text{pred}}$, then

$$L_1(\boldsymbol{y}, \boldsymbol{y}_{\text{pred}}) = \frac{1}{n} \sum_i^n \left| y^{(i)} - y_{\text{pred}}^{(i)} \right| \tag{2a}$$

$$L_2^2(\boldsymbol{y}, \boldsymbol{y}_{\text{pred}}) = \frac{1}{n} \sum_i^n \left(y^{(i)} - y_{\text{pred}}^{(i)} \right)^2 \tag{2b}$$

Equations 2a and 2b are both metrics that positively related to the magnitude of error and will be zero if no error is present between $\boldsymbol{y}$ and $\boldsymbol{y}_{\text{pred}}$. Note that Equation 2b is L2 norm square to be precise, but it is called L2 in this report for simplicity. A more interesting aspect

their gradients. Let $\nabla_{\boldsymbol{y}} L$ denote the gradient of L with respect to $\boldsymbol{y}$, then

$$\nabla_{\boldsymbol{y}} L_1 = \frac{1}{n} \cdot \text{sign}\left(\boldsymbol{y} - \boldsymbol{y}_{\text{pred}}\right) \tag{3a}$$

$$\nabla_{\boldsymbol{y}} L_2^2 = \frac{2}{n}\left(\boldsymbol{y} - \boldsymbol{y}_{\text{pred}}\right) \tag{3b}$$

We can see that the gradient of L2 is related to the error at that point while the gradient of L1 is essentially a direction which could only be either 0, 1 or -1. These characteristics can be both advantages of disadvantages. Optimizations based on L2 gradient can do updates that are more adaptive but could be harder to achieve smaller errors because the gradient would be small as well. On the other hand, Optimizations based on L1 has a more stable model update but will oscillate when the learning rate is bigger than the error.

Another way of understanding the difference between L1 and L2 loss is the priority when dealing with drastic errors or outliers. From Equation 2b we can see that L2 values the error from a point that is away from the true value a lot more than a point that is close to the truth since the error is in second-order and the gradient scales with the distance. Using an L2 norm as a loss function will focus more on solving these large errors first, which can be hard if the initialization is bad. On the other hand, L1 norm treats these errors in an equal way.

Chain rule and back-propagation

The chain rule from calculus can be summarized with the following equation. Assuming $L = f(\boldsymbol{y})$ where $\boldsymbol{y} = g(\boldsymbol{x})$, then

$$\nabla_{\boldsymbol{x}} L = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\intercal} \nabla_{\boldsymbol{y}} L \tag{4}$$

L is a scaler function that could represent the loss while $\boldsymbol{x}$ and $\boldsymbol{y}$ are two vectors that do not necessarily have the same dimension. ∇ refers to the gradient with respect to the subscript. We can see that the gradient with respect to x, $\partial \boldsymbol{y}/\partial \boldsymbol{x}$, refers to the Jacobian matrix, which links the the two gradients. One can calculate the gradient with respect to later variables ($\boldsymbol{y}$), then converted to the gradient of previous variables ($\boldsymbol{x}$). That is why this scheme is named "back-propagation". The back-propagation starts from $\partial J/\partial J = 1$ and then recursively multiply the Jacobian matrices all the way to a trainable parameter, which yields the gradient for parameter updates. Then the gradient will keep propagating until it reaches the beginning of the computation graph.

One can infer that if the chain becomes too long and any of the gradients in this chain have become a small number, the resulting gradient will end up with an even smaller number and make the parameters hard to update. This phenomenon is called vanishing gradient. There are several ways to mitigate gradient vanishing. The first cure is to avoid too deep networks. This is usually not the case because more layers are needed for adding enough non-linearity. Another popular method is to add shortcuts to the network, which allows the neural network to learn some features first and then deal with the other. U-Net (Ronneberger et al., 2015) and ResNet (He et al., 2016) are two of the most popular frameworks that use shortcuts.

Residual network (ResNet)

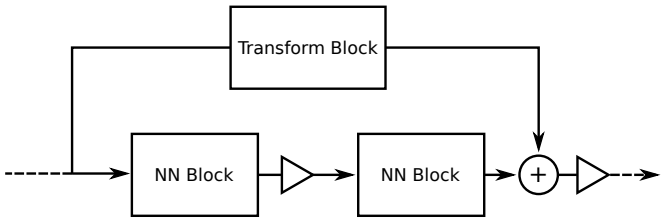

FIG. 1. A ResNet building block modified from He et al. (2016). Regular triangles refer to activation functions. Dashed arrows are connected to other blocks.

Figure 1 shows a building block of the typical ResNet. The building block contains two parts, the main stream which contains the structure of the neural networks and a shortcut to skip neural network blocks. Each of the NN blocks can contain several neural layers which can be either convolutional of fully connected. The shortcut contains a transformation that will fix the dimensional mismatch of its inputs and outputs. The transformation is linear with no activation function applied. The transformation will be identical if the input and output are already the same dimensions. Then the result of the transformation is added to the output of the main stream and the sum then feed to an activation function. The dashed line on either side may be connected to other ResNet building blocks.

The introduction of shortcuts enables the neural network to skip unnecessary steps. Since there are fewer terms when applying the chain rule. This process prevents vanishing gradient to some degree. Furthermore, the shortcuts are more than skipping some layers because of the existence of the addition node. This could be understood in another way. Suppose that the transformation block is identity. Instead of fitting a function that maps from the block input $\boldsymbol{x}$ to the block output $\boldsymbol{y}$, the ResNet block is trying to fit a function that maps from $\boldsymbol{x}$ to $(\boldsymbol{y}-\boldsymbol{x})$. In other words, the ResNet is forced to focus on learning features that are non-linear to $\boldsymbol{x}$.

EXAMPLE

Images from RTM

The velocity models are defined as 4-layer 1D models. Each of the model contains 1000 points with spacings of 8 m. We fixed the shallowest layer to have a velocity of 3000 m/s and assume it is the minimum of the entire model. The other layers have random velocities within $(3000, 5500)$ m/s. The range was designed to match the global range for the acoustic Marmousi model. The positions of reflectors are randomized with uniform distribution, however, we reject the models with layers being too thin in order to avoid severe overlapping of primaries. There is no need to apply similar restriction to the velocity differences in adjacent layer. Models can be treated as fewer layers if any adjacent velocities is close.

The RTM images are generated by following Equation 1b. The forward modelling used 2nd order finite difference method for calculating both temporal and spatial derivatives. The source wavelet is a Gaussian source. The source is non-negative and symmetric, which helps to identify reflectors. The source and receiver are both placed at 8 m below the surface. We use absorbing boundary conditions on both sides of the wavefield and direct waves are

removed before the shot record was injected for calculating the receiver wavefield.

The definition of input/output

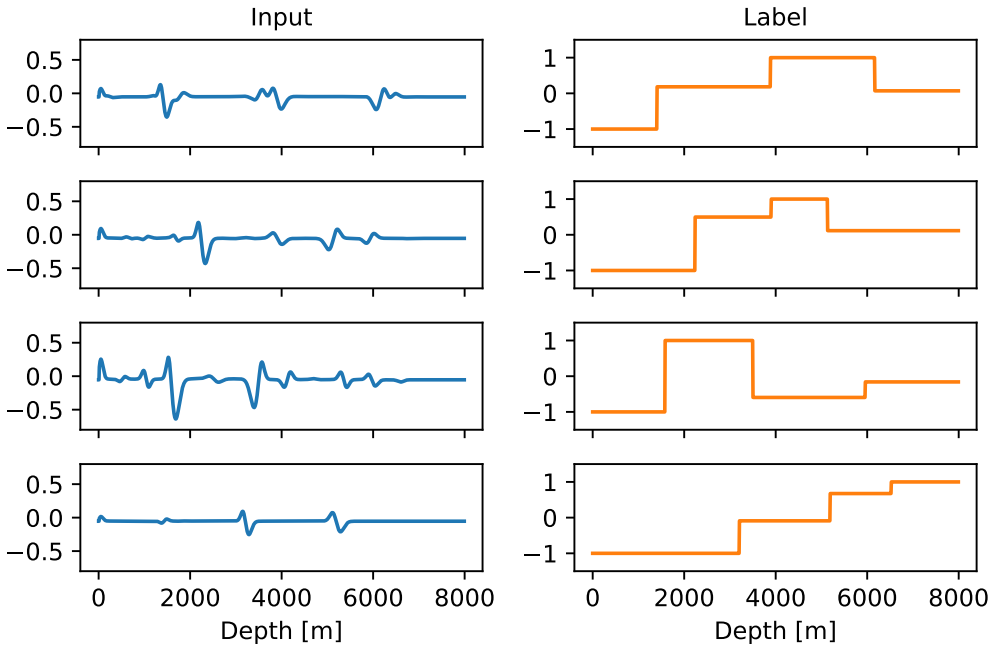

FIG. 2. Four random examples of input and label pairs.

Here we define the input fed to the neural networks to be 50000 RTM images from random 1D velocity models as described in the above section. In machine learning, the theoretical values used to calculate the loss of predictions are called true labels. In this case, the true labels are the velocity model corresponding to each image. There is an infinite number of choices on the forms of inputs and labels, which will affect the focus of the network. For example, the labels can be vectors that store respectively the depth and velocity of each layer since we have flat velocity models. The representation is efficient in terms of telling information and the model would spend no effort in learning each layer has constant velocity. The problem is that this representation makes the problem hard to generalize because we have to know the number of layers in advance and train models for different situations. Another representation is to use the true reflectivity directly as labels. This may also be problematic since the entire information is concentrated on the "spikes" in the reflectivity. Useful information will be flooded by less significant information and make the network less intuitive to what type of information should be learned.

Different input-label pairs are called *samples* in machine learning. Both input and labels have the dimensions of $n_{\text{sample}} \times n_z$, which is 50000 by 1000 in this case. Figure 2 shows 4 example input-label pairs from the dataset. The left column are inputs, which are normalized to be ranging from $(-1, 1)$. The right column shows the labels, which are the corresponding velocity models. The magnitudes have been normalized to the same range. Note that the "velocities" of the first layer are all -1. This is because they are equal to the minimum of the entire set of velocity models, i.e. 3000 m/s. For the same reason, the maximums of the labels

are all equal to 1. The entire dataset is then separated randomly into training and validation sets. The training set takes up 80% from the whole and the rest forms the validation set. The training set contains the data used directly for calculating the gradients at each iteration. The gradients would be directly related to the misfit between the predictions of the model and labels in the training set. On the other hand, the validation set is used indirectly as a metric of the optimization process. We used the performance on the validation set to help determine hyper-parameters, the degrees of over-fitting or the timing for early stopping. The trained network is then still a function of the validation set and this is how a validation set differs from the test set. In an ideal case, the dataset should be divided into training/validation/test sets. The test set, which never gets involved in the training process from the start to finish, is the only reliable metric for judging a model. However, we ignore the difference and use validation error to estimate test error in this case. This may be unfair in some sense but will allow us to have more data reserved for the training set. The ADAM optimizer (Kingma and Ba, 2014) is used throughout this report. Algorithm 1 shows a typical a training template. The algorithm saves the model that has the lowest validation error in a training process. More details about can be found in Niu and Trad (2019).

Algorithm 1 Training workflow.

Require: $\mathcal{L}(\cdot)$, model$(\cdot)$, optim$(\cdot)$
  for each epoch **do**
    for each minibatch **do**
      zero the gradients
      load X and Y ▷ load inputs/labels
      $Y_{\text{pred}} \leftarrow \text{model}(X)$ ▷ compute prediction
      $L \leftarrow \mathcal{L}(Y_{\text{pred}}, Y)$ ▷ compute loss
      $g \leftarrow BP(L)$ ▷ back-propagate
      $\text{model}(\cdot) \leftarrow \text{model}(\cdot) + \text{optim}(g)$ ▷ update model parameters
    $Y_{\text{val}} \leftarrow \text{model}(X_{\text{val}})$
    $L_{\text{val}} \leftarrow \mathcal{L}(Y_{\text{val}}, Y)$ ▷ compute validation loss
    if L_{val} is the smallest **then**
      save the model$(\cdot)$

Fully connected neural network

All neural networks in this report are implemented with the machine learning package PyTorch (Paszke et al., 2017). A fully connected model is used for solving the original problem (see Figure 3). The model contains 7 layers. The input layer and output layer have 1000 nodes, the same as the length of the model. There are 5 hidden layers in between with 500 nodes in each layer. The activation function for each hidden layer is leaky ReLU (leaky rectified linear unit) with a slope of 0.2 on the negative side. The leaky ReLu will capture features on the negative half but still add non-linearity. The activation function for the outputting layer is $\tanh$. The connections between two adjacent layers in a fully connected network can be represented by a matrix containing trainable parameters. In this network structure, the trainable parameters are, 1000 by 500 for the connections between the inputs and 1st layer; 4 of 500 by 500 matrices for interconnection between hidden layers; and a 500 by 1000 between the last hidden layer and the outputting layer. Despite the biases,

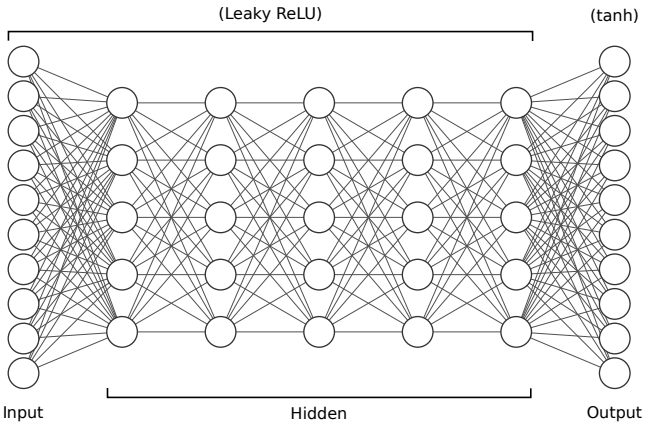

FIG. 3. A 7-layer fully connected neural network. Each circle represents 100 nodes.

the total number of trainable parameters is 2×10^6, which is smaller than the number of data points provided. However, this is not sufficient to state an over-determined problem since data points maybe not fully independent of each other.

Choosing the right loss function

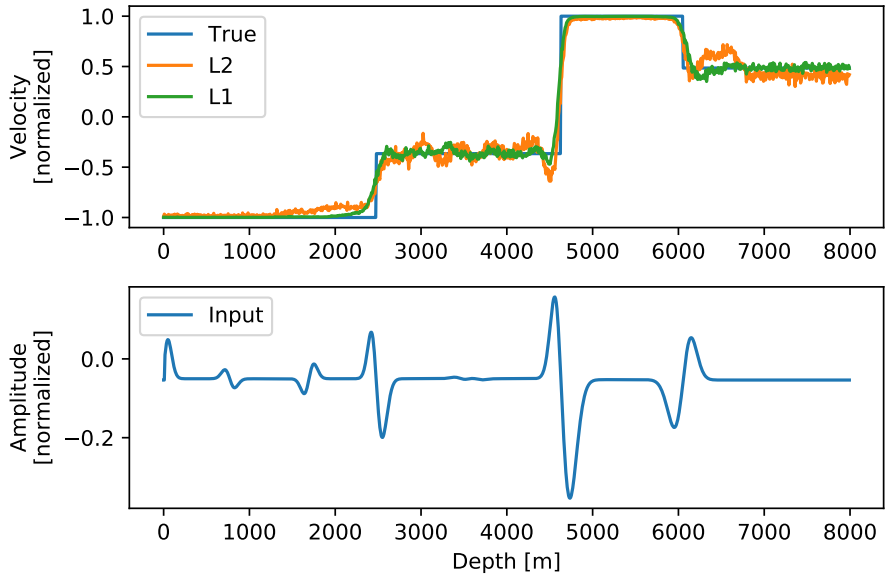

FIG. 4. Predictions made by models with L1 and L2 loss function, respectively.

Figure 4 shows predictions from different models trained with L1 and L2 loss. The bottom figure refers to the inputting RTM image and the corresponding velocity model is shown as the blue line in the top figure. We used the same network structure and hyper-parameters for both tests. We can see that the L1 prediction (green) is visually better than the L2 prediction (orange). Especially, L1 reacts faster when there is an abrupt change in the label, which eventually helps updating other velocities. We can also notice the L2 prediction is affected by another layer at around 1000 m to 1500 m. Similar observations

can be made in the last layer (from 6000 m to 6500 m), where the prediction is affected by greater velocities above it. One interesting part is that although L1 is overall more stable than L2, they have similar behaviours handling different layers. We define the model to have a fixed min and max velocity, which are -1 and 1 after normalization. This is because we use $\tanh$ as the outputting activation, which suppresses all prediction that is too large or too small. The raw output may have great oscillations.

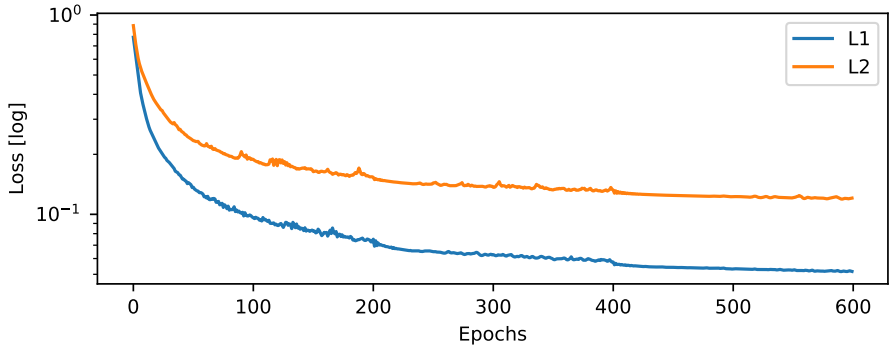

FIG. 5. L1 and L2 loss comparison.

Figure 5 shows the comparison between L1 and L2 loss on the training set. The L2 loss in Equation 2b is square rooted to be comparable with L1 loss. The figure shows that the optimization curve with L1 not only converges faster but also achieves lower error at later stage. Since the label contains abrupt changes, L2 norm focuses on dealing with those changes from the beginning but gets confused with iterations and eventually this causes fluctuations. On the other hand, L1 norm seems less affected by this issue.

Although the L1 loss curve shows that the network probably needs more iterations or a larger learning rate to reach a plateau, it proves that L1 is a more suitable loss function for this type of problem. Therefore we adopt L1 norm as loss function. However, there may be better choices of loss function, like total variation (Anagaw and Sacchi, 2012).

ResNet

Figure 6 shows the ResNet tested in this report. In addition to the network in Figure 3, skipping connections are added to the hidden layers. As shown by the black arrows, each connection skips two trainable FC layers and add the input directly to the output of the ResNet building block. Since the input and output of each building block are the same, the transformation block is identical. There are no additional trainable parameters introduced, hence the network has the same training burden.

We train the ResNet with the same setup and hyper-parameters. The loss curve on the validation set is very similar to and almost overlapping the loss curve of the fully connected case. (see Figure 7). Both models achieve small L1 errors and do a good job on identifying reflection interfaces. However, the two models make predictions differently.

As shown in Figure 8, although the two networks have similar errors, the ResNet predictions (green in the top figure) has fewer fluctuations than the fully connected (FC)

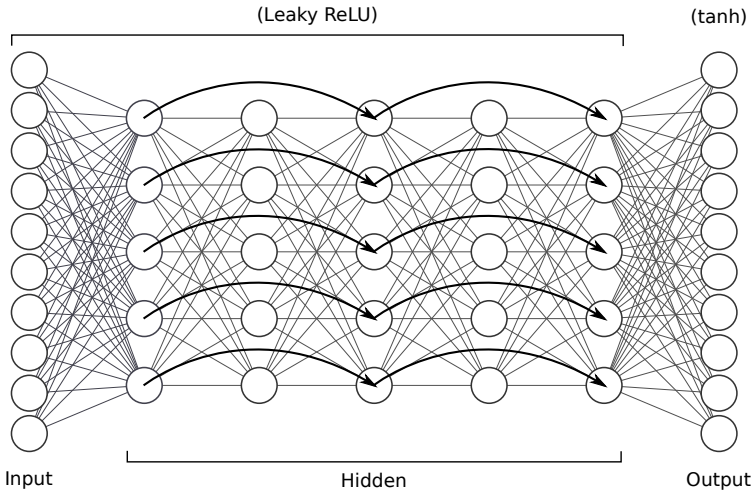

FIG. 6. A ResNet based on the fully connected network in Figure 3.

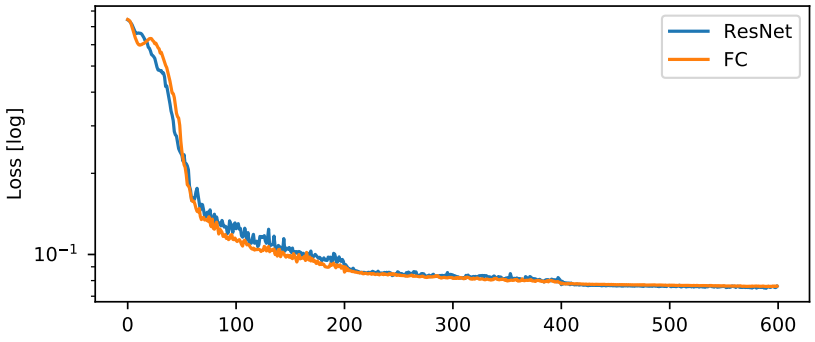

FIG. 7. Loss curves of the fully connected (FC) and the ResNet model.

predictions (orange in the top figure). This characteristic is common in different samples. The only difference between the ResNet and the FC models is the shortcuts that fed back to the main stream. The direct input from several layers before makes the network easier to find relationships between points and hence reduce the fluctuations. However, the shortcuts do not help too much on the convergence in this case. This is because we are using a relatively shallow network that may not suffer too much from vanishing gradient.

Problematic cases

For testing, one thing to keep in mind is that the testing dataset must be normalized in exactly the same way as the training dataset. In this report, the training data are normalized by linearly stretching the min and max value to $(-1, 1)$. The test dataset must be stretched in the same way—with the min and max of the training dataset but not its own.

As shown in Figure 9, We test the model with inputs and output pairs generated by using a different wavelet (Ricker wavelet). The image (bottom) is easy for human brains, especially for recognizing the positions of the reflectors. However, the predictions are bad

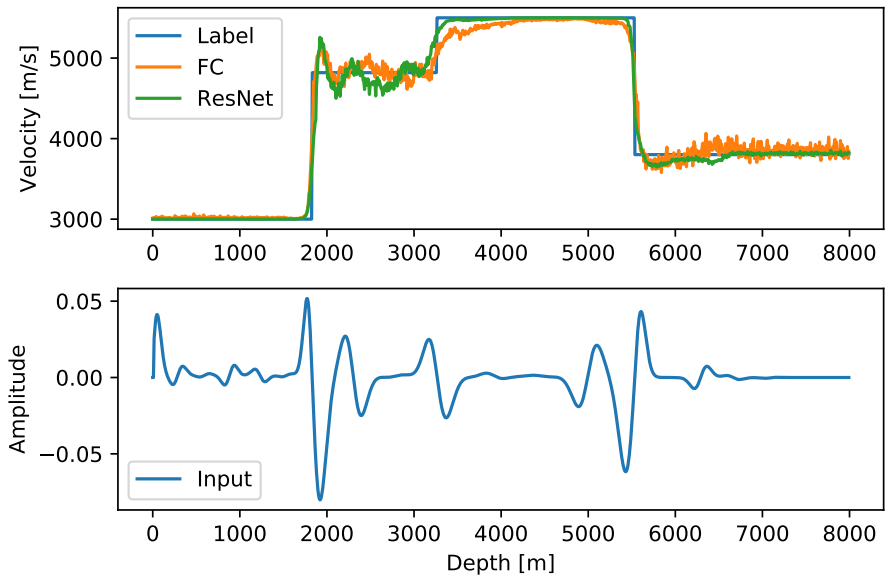

FIG. 8. A comparison between predictions from the fully connected (FC) model and the ResNet.

for both models. The predictions somewhat react to reflections at the first two interfaces but failed to detect the deepest interface. Also, the velocity is correct only for the first 150 points and this is partly because the velocity of the first layer is fixed. Similar results can be observed if we change the imaging condition. This is certain because we have broken the fundamental rule of machine learning: the test set must come from the same distribution for the training set. In order to make the model work with different wavelets, we should either remove the wavelet effect by some methods (as we remove dependencies on acquisitions by migrations) or provide enough data for the network to learn about the change. The former will make the whole problem less meaningful as if the wavelet effect is fully removed, the results will be the true reflectivity and we can get velocity by integration. The latter would require the dataset to be several times larger and perhaps need a network with more complex structure, more trainable parameters and more advanced technique for the training (such as gradient boosting, which takes a lot more power to perform). This is hard to do, either more data is not available or computation cost is too high even for today's computation power.

CONCLUSION

In this report, we use fully-connected networks to recover the migration images from random four 4-layer velocity models. We investigate different behaviours when using L1/L2 norms as loss function and we conclude L1 is more suitable for this type of problem. We test ResNet shortcuts to the network and they reduce fluctuations. The model performs poorly on data from different distributions of the training set. Future works may include applying more advance training techniques like gradient boosting or seeking better representations of the input and outputs. Also, we may need to try total variation instead of L1 norm.

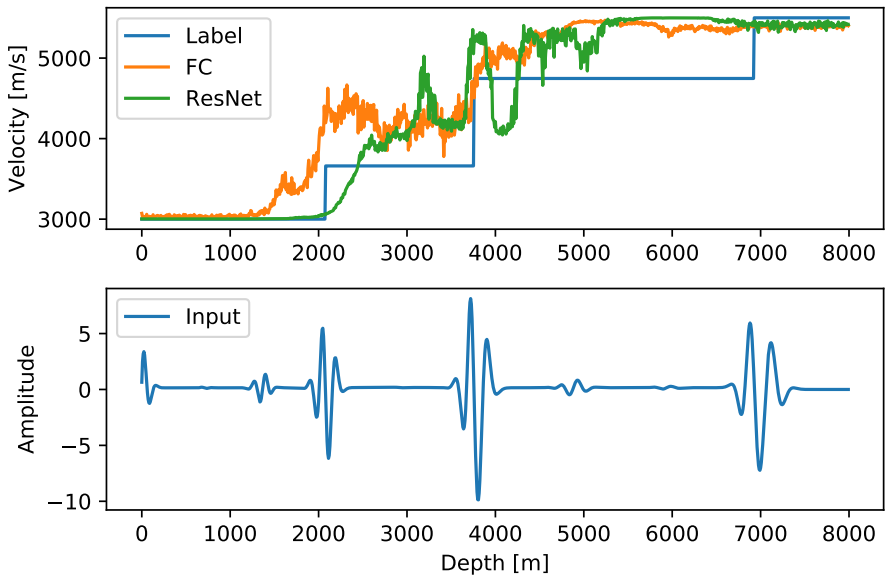

FIG. 9. A typical prediction on data with Ricker wavelet.

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. Special thanks to Jian Sun at Penn State University and Marcelo Guarido for valuable discussions.

REFERENCES

Anagaw, A. Y., and Sacchi, M. D., 2012, Edge-preserving seismic imaging using the total variation method: Journal of Geophysics and Engineering, **9**, No. 2, 138–146.

Claerbout, J. F., 1971, Toward a unified theory of reflector mapping: Geophysics, **36**, No. 3, 467–481.

He, K., Zhang, X., Ren, S., and Sun, J., 2016, Deep residual learning for image recognition, *in* Proceedings of the IEEE conference on computer vision and pattern recognition, 770–778.

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.

Niu, Z., and Trad, D., 2019, Deblending using convolutional neural networks: CREWES Research Report.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A., 2017, Automatic differentiation in PyTorch, *in* NIPS Autodiff Workshop.

Ronneberger, O., Fischer, P., and Brox, T., 2015, U-net: Convolutional networks for biomedical image segmentation, *in* International Conference on Medical image computing and computer-assisted intervention, Springer, 234–241.

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

Taylor, H. L., Banks, S. C., and McCoy, J. F., 1979, Deconvolution with the l1 norm: Geophysics, **44**, No. 1, 39–52.