Machine learning and geophysical inversion — A numerical study

Brian Russell[1]

https://doi.org/10.1190/tle38070512.1.

Abstract

As geophysicists, we are trained to conceptualize geophysical problems in detail. However, machine learning algorithms are more difficult to understand and are often thought of as simply "black boxes." A numerical example is used here to illustrate the difference between geophysical inversion and inversion by machine learning. In doing so, an attempt is made to demystify machine learning algorithms and show that, like inverse problems, they have a definite mathematical structure that can be written down and understood. The example used is the extraction of the underlying reflection coefficients from a synthetic seismic response that was created by convolving a reflection coefficient dipole with a symmetric wavelet. Because the dipole is below the seismic tuning frequency, the overlapping wavelets create both an amplitude increase and extra nonphysical reflection coefficients in the synthetic seismic data. This is a common problem in real seismic data. In discussing the solution to this problem, the topics of deconvolution, recursive inversion, linear regression, and nonlinear regression using a feedforward neural network are covered. It is shown that if the inputs to the deconvolution problem are fully understood, this is the optimal way to extract the true reflection coefficients. However, if the geophysics is not fully understood and large amounts of data are available, machine learning can provide a viable alternative to geophysical inversion.

Introduction

Several recent studies (Araya-Polo et al., 2018; Kim and Nakata, 2018; Lu et al., 2018; Naeini and Prindle, 2018) apply machine learning algorithms to geophysical inversion and interpretation problems. In particular, Kim and Nakata (2018) apply both deconvolution and a deep neural network to the estimation of seismic reflectivity from both a synthetic wedge model and recorded seismic data. The key geophysical problem addressed by Kim and Nakata (2018) is wavelet tuning in a wedge model, described in a classic paper by Widess (1973). The authors conclude that machine learning is able to solve the wavelet-tuning problem more exactly than the deconvolution method can. In what follows, the wavelet-tuning problem is isolated and analyzed on a small part of a single seismic trace, using both deconvolution and a single hidden-layer neural network. The results show that, if we have enough knowledge about the wavelet and the geology, deconvolution is the preferred solution. However, in the case of less-than-perfect knowledge of the inputs to deconvolution, machine learning with neural networks can perform just as well as, or even better than, deconvolution.

The seismic model

The noise-free normal incidence acoustic convolutional model is written as

$$\boldsymbol{s} = G\boldsymbol{r}, \tag{1}$$

where $\boldsymbol{s}$ is a vector with n seismic trace samples, G is an n by m dimensional geophysical transform matrix that contains a shifted seismic wavelet in each of its columns, and $\boldsymbol{r}$ is a vector with m zero-offset reflection coefficients (Claerbout, 1976). For $m + 1$ acoustic earth layers, the m reflection coefficients are given by

$$r_i = \frac{\rho_{i+1}V_{i+1} - \rho_i V_i}{\rho_{i+1}V_{i+1} + \rho_i V_i} = \frac{I_{i+1} - I_i}{I_{i+1} + I_i}, \tag{2}$$

where $i = 1,\ldots, m$, ρ_i is the density, V_i is the P-wave velocity, and I_i is the P-wave impedance of each layer. Equation 2 can be extended to the elastic nonnormal incidence case using either the nonlinear Zoeppritz equations or the linearized Aki-Richards equations, where the reflection coefficients vary as a function of angle of incidence, and three elastic constants — density, P-wave velocity, and S-wave velocity — are required for each layer.

If we are able to identify the reflection coefficients in equation 2, seismic inversion is straightforward and consists of the recursive formula given by

$$I_{i+1} = I_i \frac{1 + r_i}{1 - r_i}, \tag{3}$$

where $i = 1, 2,\ldots, m$.

To illustrate equations 2 and 3, consider the three-layer earth model shown in Figure 1, which consists of a shale of P-impedance 4500 m/s*g/cc that both overlies and underlies a wet sandstone of P-impedance 5500 m/s*g/cc (i.e., $m + 1 = 3$ and there are $m = 2$ reflection coefficients). Application of equation 2 gives

$$\boldsymbol{r} = \begin{bmatrix} +0.1 \\ -0.1 \end{bmatrix}, \tag{4}$$

which is often called a dipole reflectivity, as shown in Figure 1.

Using the reflectivity calculated in equation 4 and the correct initial guess of 4500 m/s*g/cc, application of the recursive inversion approach given in equation 3 gives the correct answer of

$$\boldsymbol{I} = \begin{bmatrix} 4500 \\ 5500 \\ 4500 \end{bmatrix}. \tag{5}$$

[1]CGG GeoSoftware, Calgary, Alberta, Canada. E-mail: brian.russell@cgg.com.

To create a synthetic seismic trace model using equation 1, the key consideration is the thickness of the sandstone layer with respect to the seismic wavelength. The thickness can be varied using the wedge model shown in Figure 2. In this model, the well log shown in Figure 1 is inserted at trace 25, and the wedge is thinned from 200 ms at trace 1 to 0 ms at trace 50. A 30 Hz Ricker wavelet has been used to create each synthetic trace.

In Figure 2, two independent wavelets are visible until the wavelets start to overlap and form a single 90° phase wavelet. This is called wavelet tuning (Widess, 1973). Notice in Figure 2 that the amplitudes of both the peak and trough increase as the events start to tune together to the right of common midpoint (CMP) 44. Also, there is a time "pull-up" due to the fact that the zero crossing of the 90° wavelet is centered at the pinchout time of 300 ms. Finally, the wavelets cancel completely at CMP 50.

To illustrate the effect of wavelet tuning, a seismic trace is created by convolving the dipole reflectivity of equation 4 with a three-point wavelet given by

$$\boldsymbol{g} = \begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix}. \tag{6}$$

To produce full wavelet tuning, the wavelet is shifted by a single sample, giving

$$\boldsymbol{s} = G\boldsymbol{r} = \begin{bmatrix} -1 & 0 \\ 2 & -1 \\ -1 & 2 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} +0.1 \\ -0.1 \end{bmatrix} = \begin{bmatrix} -0.1 \\ +0.3 \\ -0.3 \\ +0.1 \end{bmatrix}. \tag{7}$$

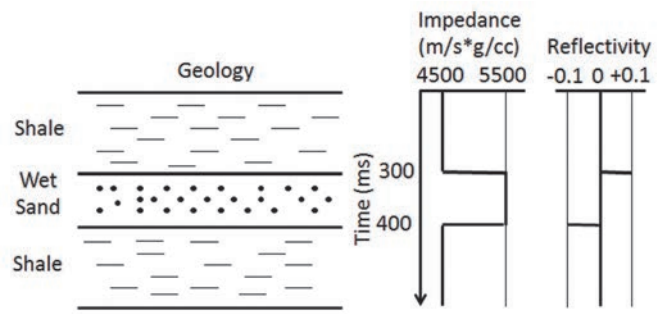

Figure 1. A three-layer earth model, integrated to seismic time and showing both the impedances and reflection coefficients from equations 4 and 5.

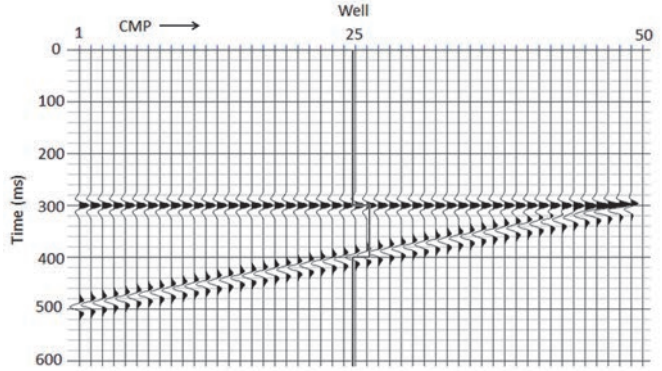

Figure 2. A wedge model that uses the three layers shown in Figure 1.

In equation 7, each of the two columns in G represents the discrete wavelet of equation 6 shifted by one sample, and thus the multiplication of G with $\boldsymbol{r}$ is equivalent to the convolution of wavelet $\boldsymbol{g}$ with $\boldsymbol{r}$. In this case, the number of seismic samples (n = 4) is the sum of the length of the wavelet and the one sample shift. Equation 7 produces the classic tuned seismic response and displays both an amplitude increase compared with the well-log reflectivity and the creation of an apparent 90° wavelet. Inverting the synthetic response of equation 7 using equation 3, and starting with the correct impedance, gives

$$\hat{\boldsymbol{I}}^T = \begin{bmatrix} 4500 & 3682 & 6838 & 3682 & 4500 \end{bmatrix}, \tag{8}$$

where the "hat" over $\boldsymbol{I}$ indicates that this is an estimate of the true impedance, and the superscript T indicates the transpose of the vector from column to row format. Notice that two "pseudolayers" have been added to the true impedance due to the wavelet side lobes. Also, the increased amplitudes lead to a very large impedance value in the third layer.

Deconvolution

If we are able to correctly estimate both the exact shape of the wavelet and the exact spacing of the reflection coefficients (i.e., we know both G and $\boldsymbol{s}$ exactly), we can use the technique of deconvolution to exactly recover the reflectivity. If the wavelet matrix was square, we could solve the problem by a straightforward matrix inversion as

$$\boldsymbol{r} = G^{-1}\boldsymbol{s}. \tag{9}$$

But because the problem is overdetermined (i.e., more equations than unknowns) we first multiply both sides of equation 9 by the transpose of G, giving

$$G^T\boldsymbol{s} = (G^T G)\boldsymbol{r}. \tag{10}$$

In equation 10, the left-hand side is the crosscorrelation of the wavelet with the seismic trace ($G^T\boldsymbol{s}$), and the right-hand side is the autocorrelation of the wavelet matrix ($G^T G$) multiplied by the reflectivity. Equation 10 is now a square matrix equation and can be inverted to give

$$\boldsymbol{r} \approx (G^T G + \lambda I)^{-1} G^T\boldsymbol{s} = G^*\boldsymbol{s}, \tag{11}$$

where $G^* = (G^T G + \lambda I)^{-1} G^T$ is the generalized inverse of the wavelet matrix, and a prewhitening factor λI (where I is the 2 × 2 identity matrix) has been added to avoid instability in the inversion. Inserting the numerical values from equation 7 into equation 11, and using a prewhitening factor of zero, gives

$$\boldsymbol{r} = G^*\boldsymbol{s} = \begin{bmatrix} -0.3 & 0.4 & 0.1 & -0.2 \\ -0.2 & 0.1 & 0.4 & -0.3 \end{bmatrix} \begin{bmatrix} -0.1 \\ +0.3 \\ -0.3 \\ +0.1 \end{bmatrix} = \begin{bmatrix} +0.1 \\ -0.1 \end{bmatrix}. \tag{12}$$

In this ideal case, the true reflectivity is recovered exactly. When dealing with real data where the data are noisy and we

know neither the wavelet nor the positions of the reflectors, the solution is not nearly as good and results in a band-limited reflectivity estimate.

Linear regression

In the previous section, we considered the case in which the seismic trace and wavelet matrix were known. In this section, we assume that the reflectivity and seismic trace are both known, and we wish to derive a relationship between the two. The simplest solution to this problem is linear regression, where we estimate the two unknown weights w_0 (the intercept or bias) and w_1 (the slope) in the equation

$$\boldsymbol{r} = w_0 + w_1\boldsymbol{s}. \tag{13}$$

Similar to the deconvolution problem, linear regression can be written in matrix format as follows, where the reflectivity has been padded with zeros to make it the same length as the seismic trace:

$$\boldsymbol{r} = S\boldsymbol{w} = \begin{bmatrix} 1 & -0.1 \\ 1 & +0.3 \\ 1 & -0.3 \\ 1 & +0.1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} 0 \\ +0.1 \\ -0.1 \\ 0 \end{bmatrix}. \tag{14}$$

In equation 14, the column of ones in the S matrix is there to multiply the bias term w_0. Although equation 14 is almost identical to equation 7, the unknowns are now the weights rather than the reflection coefficients. As in the deconvolution problem, both sides of equation 14 are first multiplied by the transpose of S to create the square matrix equation given by

$$S^{\mathrm{T}}\boldsymbol{r} = (S^{\mathrm{T}}S)\,\boldsymbol{w}. \tag{15}$$

The inversion of equation 15 to give the least-squares solution of the weights is given as

$$\boldsymbol{w} = (S^{\mathrm{T}}S + \lambda I)^{-1}\,S^{\mathrm{T}}\boldsymbol{r} = S^{*}\boldsymbol{r}, \tag{16}$$

where $S^{*} = (S^{\mathrm{T}}S + \lambda I)^{-1}\,S^{T}$ is the generalized inverse of the seismic matrix, and λ is again a prewhitening factor. Using our model values with a prewhitening factor of zero gives

$$\boldsymbol{w} = S^{*}\boldsymbol{r} = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ -0.5 & 1.5 & -1.5 & 0.5 \end{bmatrix} \begin{bmatrix} 0 \\ +0.1 \\ -0.1 \\ 0 \end{bmatrix} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.3 \end{bmatrix}. \tag{17}$$

Equation 17 estimates a bias ($\boldsymbol{w}_0$) of zero since the seismic trace and reflectivity both have zero mean. The second weight is simply a scaling coefficient that attempts to match the amplitudes between the seismic and reflectivity, and can be applied as

$$\hat{\boldsymbol{r}} = w_1\boldsymbol{s} = 0.3\begin{bmatrix} -0.1 \\ +0.3 \\ -0.3 \\ +0.1 \end{bmatrix} = \begin{bmatrix} -0.03 \\ +0.09 \\ -0.09 \\ +0.03 \end{bmatrix}, \tag{18}$$

where the hat over the reflectivity indicates that this is an estimate of the true reflectivity. The estimate shown in equation 18 is closer to the correct reflectivity than the unscaled reflectivity, but the two true reflection coefficients have been slightly underscaled. Applying the inversion formula of equation 3 gives

$$\hat{\boldsymbol{I}}^{T} = \begin{bmatrix} 4500 & 4238 & 5076 & 4238 & 4500 \end{bmatrix}. \tag{19}$$

The solution shown in equation 19 is closer to the correct answer than the inversion of the seismic trace, but it is still incorrect because of the wavelet side lobes and incorrect scaling. Figure 3 is a display of the inversion of both the seismic reflectivity and the scaled seismic reflectivity compared to the true impedances.

Another way to visualize the weights in least-squares regression is to use them to fit a continuous straight line to reflection coefficients versus seismic amplitudes. This is shown in Figure 4, where the true values are shown by the black points. The line in Figure 4 represents the equation $\hat{\boldsymbol{r}} = 0 + 0.3\boldsymbol{s}$ and illustrates the difference between this approach and the previous deconvolution approach. In deconvolution, we got a perfect fit to the four points because our assumptions about the seismic model were correct. In least-squares regression, the points have been fit in a "best" least-squares sense, which is not exact.

Gradient descent methods

In both the deconvolution and regression methods discussed previously, the full matrix was inverted. For the size of data sets normally used in seismic analysis, this is impractical, and we would normally use iterative techniques that do not involve calculating a matrix inverse. The simplest iterative technique is called steepest descent (SD), in which we iteratively arrive at a solution by starting with an initial guess. The SD algorithm for regression is written as

$$\boldsymbol{w}_{(k+1)} = \boldsymbol{w}_{(k)} + \alpha_{(k)}\boldsymbol{\delta}_{(k)}, \tag{20}$$

where $k = 0, \ldots, K$, $\boldsymbol{w}_{(k+1)}$ is the vector of weights at the $k + 1^{\mathrm{st}}$ iteration, $\boldsymbol{\delta}_{(k)} = (S^{T}S)\boldsymbol{w}_{(k)} - S^{T}\boldsymbol{r}$ is the error found for the k^{th} application of the algorithm and $\alpha_{(k)}$ is the learning rate. Note that the term $\boldsymbol{\delta}_{(k)}$ is simply the difference between the right and left sides of equation 15 at the k^{th} update of the weights. To start the SD algorithm, we require an initial estimate of the weights, which is usually a random guess.

A more advanced iterative technique is the conjugate gradient (CG) algorithm (Shewchuck, 1994; Schleicher, 2018), where the algorithm takes steps that are orthogonal to the change in gradient, rather than the gradient itself. A variant of the SD algorithm is called the least-mean-square (LMS) algorithm (Widrow and Hoff, 1960), which has applications in heart

monitoring and noise-cancelling headphones. In the LMS algorithm, the weights are trained one sample at a time, and thus the method is time adaptive. In neural network applications, the LMS algorithm is called stochastic gradient descent.

A comparison of the SD, LMS, and CG algorithms applied to our regression example is shown in Figure 5. The SD algorithm, shown as the dotted line, moves from an initial guess of (1, 1) to the true answer of (0, 0.3) in steps that are orthogonal to each other. The CG algorithm, shown as the dashed line, converges to the correct answer in two steps. (For linear problems, it can be shown that the CG algorithm always converges in the same number of steps as the number of unknown weights). As seen in Figure 5, the first step in CG is identical to the first step in SD. The α term in both SD and CG was optimized using line minimization (Hagan et al., 1996) and varied between 0.05 and 0.2. Finally, the LMS algorithm, shown by the solid line, required 10,000 iterations to converge to an answer with a much smaller α term of 0.001, which was held fixed. The contours shown in Figure 5 represent the performance surface for this problem, which is an ellipsoidal surface with the correct weights at the minimum point on the surface.

The feedforward neural network and nonlinear regression

In Figure 4, it was noted that the straight-line solution given by linear regression did not give a perfect fit between the true seismic and reflectivity values. Neural networks, which are a type of machine learning algorithm, allow us to extend linear regression to nonlinear regression. The neural network used here has two different names that at first seem contradictory: the feedforward neural network and the backpropagation neural network. The term feedforward refers to how the output is computed from the input if the weights have already been determined. The term backpropagation refers to how the training of the weights is performed, using a technique called error backpropagation.

Figure 6 shows a flow diagram of a three-layer neural network. The first layer is called the input layer and consists of the vector of seismic values ($\boldsymbol{s}$) and a vector of ones of the same length ($\mathbf{1}$). Equivalently, we could combine the seismic vector and the ones vector into the matrix S, as shown in equation 14. This is called batch input since we input all the recorded samples at once. Alternately, we could input the samples one-by-one in real time, which is called stochastic input. In this study we use batch input.

The input values are now multiplied by the two sets of weights and output the two intermediate vectors given by

$$\boldsymbol{y}_1^{(1)} = w_{01}^{(1)}\mathbf{1} + w_{11}^{(1)}\boldsymbol{s} \tag{21}$$

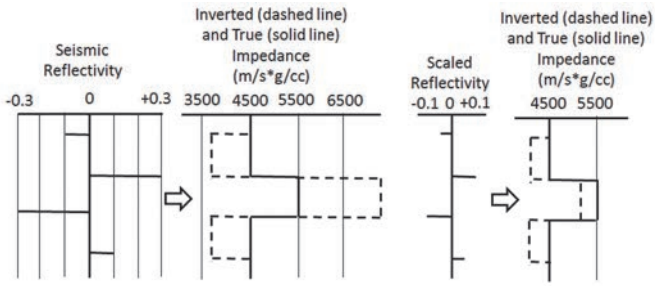

Figure 3. The inversion of the tuned seismic reflectivity on the left and scaled reflectivity on the right (dashed lines) compared to the inversion of the true reflectivity (solid line).

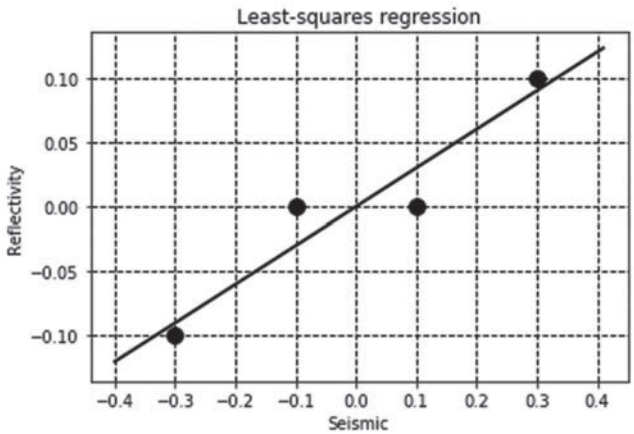

Figure 4. Linear regression between the reflectivity and seismic amplitudes using the least-squares regression coefficients of equation 17.

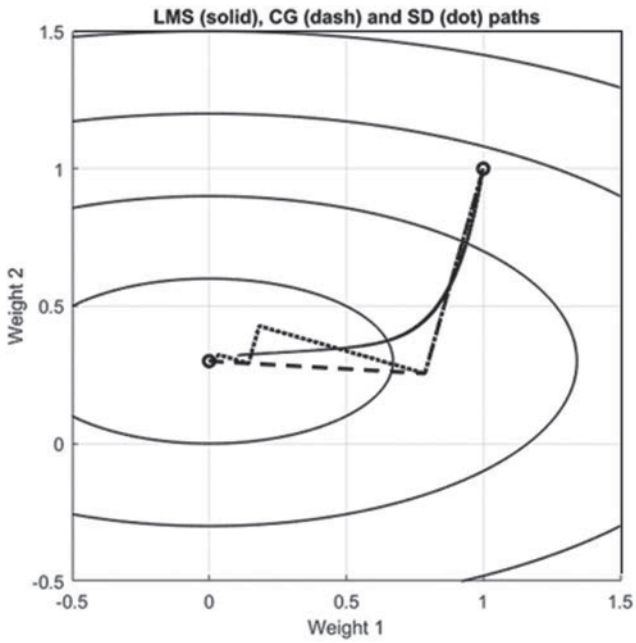

Figure 5. A comparison of the LMS (solid line), CG (dashed line), and SD (dotted line) algorithms for solving the linear regression problem, where the contours represent the performance surface.

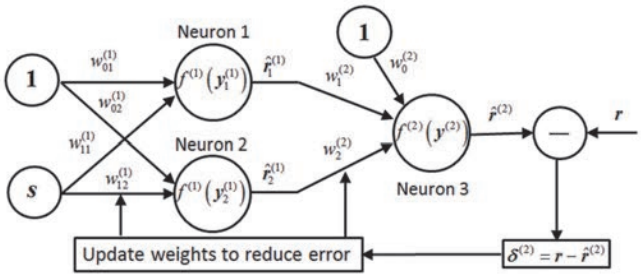

Figure 6. A feedforward neural network with a single hidden layer.

and

$$\boldsymbol{y}_2^{(1)} = w_{02}^{(1)}\mathbf{1} + w_{12}^{(1)}\boldsymbol{s}\,, \tag{22}$$

where $\mathbf{1}^T = [1\ \ 1\ \ 1\ \ 1]$, $\mathbf{s}^T = [s_1\ \ s_2\ \ s_3\ \ s_4]$, and the superscript (1) indicates that this is the input to the first hidden layer. The term hidden just means that the calculation of the weight values is done internally by the network and is thus hidden to us. The two weights $w_{01}^{(1)}$ and $w_{02}^{(1)}$ are referred to as the bias weights, since they apply a DC bias to the inputs to each neuron. We can also rewrite equations 21 and 22 as the single equation given by

$$Y^{(1)} = SW^{(1)} = \begin{bmatrix} 1 & -0.1 \\ 1 & +0.3 \\ 1 & -0.3 \\ 1 & +0.1 \end{bmatrix} \begin{bmatrix} w_{01}^{(1)} & w_{02}^{(1)} \\ w_{11}^{(1)} & w_{12}^{(1)} \end{bmatrix}. \tag{23}$$

The S matrix in equation 23 is the same as in the linear regression equation 14, but the weights from equations 21 and 22 are now arranged in the matrix $W^{(1)}$, which creates the two vectors shown in equations 21 and 22 as the columns of the matrix $Y^{(1)}$.

The key difference between neural networks and linear regression is that each neuron applies a linear or nonlinear function to the weighted sums $\boldsymbol{y}_1^{(1)}$ and $\boldsymbol{y}_2^{(1)}$, as shown in Figure 6. A common nonlinear function is the logistic function given by

$$f(y) = \frac{1}{1+\exp(-y)}\,. \tag{24}$$

The logistic function is used for two reasons. Its graphical form, shown in Figure 7, is sigmoidal, or S-shaped, and the function therefore moves smoothly between limits of 0 and 1.

Also, the logistic function has the very simple derivative given by

$$\frac{df(y)}{dy} = f(y)\big(1-f(y)\big), \tag{25}$$

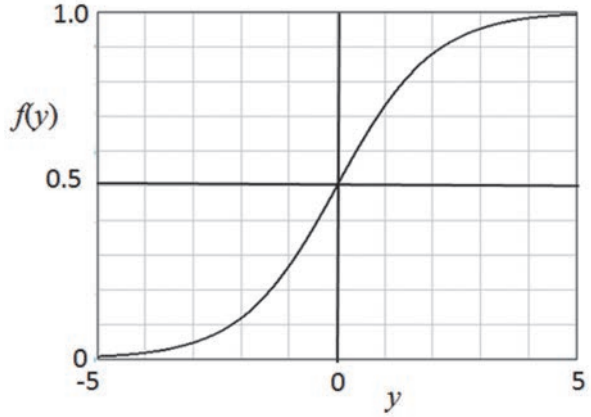

Figure 7. The logistic function defined by equation 24.

which simplifies the calculations in the backpropagation algorithm. In the neural network shown in Figure 6, the function $f^{(1)}(y)$ is the logistic function and the function $f^{(2)}(y)$ is the linear function y, meaning that its derivative is simply unity.

Applying the logistic function in the neural network flowchart of Figure 6 then gives two intermediate estimates of the reflectivity given by

$$\hat{\boldsymbol{r}}_1^{(1)} = \frac{1}{1+\exp\left(-\boldsymbol{y}_1^{(1)}\right)} \tag{26}$$

and

$$\hat{\boldsymbol{r}}_2^{(1)} = \frac{1}{1+\exp\left(-\boldsymbol{y}_2^{(1)}\right)}. \tag{27}$$

Equations 26 and 27 can be expressed in matrix format as

$$R^{(1)} = F^{(1)}\left(Y^{(1)}\right) = \begin{bmatrix} 1 & \hat{r}_{11}^{(1)} & \hat{r}_{12}^{(1)} \\ 1 & \hat{r}_{21}^{(1)} & \hat{r}_{22}^{(1)} \\ 1 & \hat{r}_{31}^{(1)} & \hat{r}_{32}^{(1)} \\ 1 & \hat{r}_{41}^{(1)} & \hat{r}_{42}^{(1)} \end{bmatrix}, \tag{28}$$

where $\hat{r}_{ij}^{(1)} = \dfrac{1}{1+\exp\left(-y_{ij}^{(1)}\right)}$. In equation 28, a column of ones has been inserted into the matrix $R^{(1)}$ to multiply by the weight bias in the final step.

The last part of the neural network of Figure 6 is called the output layer and consists of inputting the weighted sum of the outputs of neurons 1 and 2 to neuron 3. The weighted sum can be written in vector format as

$$\hat{\boldsymbol{y}}^{(2)} = w_0^{(2)}\mathbf{1} + w_1^{(2)}\hat{\boldsymbol{r}}_1^{(1)} + w_2^{(2)}\hat{\boldsymbol{r}}_2^{(1)}\,, \tag{29}$$

where $\mathbf{1}^T = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$, $\hat{\boldsymbol{r}}_1^{(1)T} = \begin{bmatrix} \hat{r}_{11}^{(1)} & \hat{r}_{21}^{(1)} & \hat{r}_{31}^{(1)} & \hat{r}_{41}^{(1)} \end{bmatrix}$, $\hat{\boldsymbol{r}}_2^{(1)T} = \begin{bmatrix} \hat{r}_{12}^{(1)} & \hat{r}_{22}^{(1)} & \hat{r}_{32}^{(1)} & \hat{r}_{42}^{(1)} \end{bmatrix}$, and $w_0^{(2)}$ is the output bias weight. The final step is to apply the second function $f^{(2)}$ to the weighted input, which gives:

$$\hat{\boldsymbol{r}}^{(2)} = f^{(2)}\left(w_0^{(2)}\mathbf{1} + w_1^{(2)}\hat{\boldsymbol{r}}_1^{(1)} + w_2^{(2)}\hat{\boldsymbol{r}}_2^{(1)}\right) = w_0^{(2)}\mathbf{1} + w_1^{(2)}\hat{\boldsymbol{r}}_1^{(1)} + w_2^{(2)}\hat{\boldsymbol{r}}_2^{(1)}, \tag{30}$$

where the final estimate of the reflectivity is equal to $\hat{\boldsymbol{y}}^{(2)}$ since the second function is linear. We can write this in vector format as

$$\hat{\boldsymbol{r}}^{(2)} = \hat{\boldsymbol{y}}^{(2)} = R^{(1)}\boldsymbol{w}^{(2)}\,, \tag{31}$$

where $\boldsymbol{w}^{(2)T} = \begin{bmatrix} w_0^{(2)} & w_1^{(2)} & w_2^{(2)} \end{bmatrix}$.

Backpropagation

To find the optimum weights for our neural network, a procedure called error backpropagation (Rummelhart et al., 1986) is used. The algorithm is written as follows:

- initialize the weights in both layers to small random values;
- starting with the weights in the output layer, change the weights to minimize the error between the computed and desired output values;
- backpropagate the error minimization for all layers;
- iterate until an acceptable error is found.

Referring back to Figure 6, the final error is given by the difference between the desired output and the second-layer output, and is written as

$$\boldsymbol{\delta}^{(2)} = \boldsymbol{r} - \hat{\boldsymbol{r}}^{(2)}. \tag{32}$$

This error is then used to update the weights so that the error is minimized in a least-squares sense. The main concept behind the error minimization is taken from the chain rule of calculus and involves differentiating the functions in the neural network. For a detailed derivation of backpropagation, see Hagan et al. (1996).

Backpropagation starts by iteratively updating the weights in the second layer. For the $k + 1^{st}$ iteration, we have

$$\boldsymbol{w}^{(2)}_{(k+1)} = \boldsymbol{w}^{(2)}_{(k)} + \alpha R^{(1)T}_{(k)} \boldsymbol{\delta}^{(2)}_{(k)}, \tag{33}$$

where $k = 0,\ldots, K$, $\boldsymbol{\delta}^{(2)}_{(k)}$ is the final error as given in equation 32, $R^{(1)T}_{(k)}$ is the matrix transpose of the output of the hidden layer as given in equation 28, and α is the learning rate, which in this case is fixed. With the exception of the matrix $R^{(1)T}_{(k)}$, note the similarity of equation 33 to the SD equation (equation 20). Since the second-layer function is linear, its derivative is equal to unity and so does not appear in equation 30. The second step is to update the weights in layer 1 as follows

$$W^{(1)T}_{(k+1)} = W^{(1)T}_{(k)} + \alpha S \boldsymbol{\delta}^{(1)}_{(k)}, \tag{34}$$

where $\boldsymbol{\delta}^{(1)}_{(k)} = \left[R^{(1)}_{(k)} \circ (1 - R^{(1)}_{(k)})\right]^T \circ \left[\boldsymbol{w}^{(2)}_{(k+1)} \boldsymbol{\delta}^{(2)T}_{(k+1)}\right]$ The symbol $\circ$ in the computation of $\boldsymbol{\delta}^{(1)}_{(k)}$ implies an element-by-element multiplication of two matrices rather than a standard matrix multiplication. Also, note that the first term in the calculation is the derivative of $R^{(1)T}_{(k)}$ as given by equation 25. Due to the extra column of ones that had been added earlier to multiply with the bias weights, there is a column of zeros in the second term on the right-hand side of equation 34 that must be removed. Some authors (e.g., Hagan et al., 1996) get around this by treating the bias terms separately.

Backpropagation starts with a random guess of the initial weights at step $k = 0$. Taking values from a normal distribution that ranges between –1 and +1, our initial weights are

$$W^{(1)}_{(0)} = \begin{bmatrix} w^{(1)}_{01(0)} & w^{(1)}_{02(0)} \\ w^{(1)}_{11(0)} & w^{(1)}_{12(0)} \end{bmatrix} = \begin{bmatrix} 0.0940 & 0.4894 \\ -0.4074 & -0.6221 \end{bmatrix} \tag{35}$$

and

$$\boldsymbol{w}^{(2)}_{(0)} = \begin{bmatrix} w^{(2)}_{0(0)} \\ w^{(2)}_{1(0)} \\ w^{(2)}_{2(0)} \end{bmatrix} = \begin{bmatrix} 0.3736 \\ -0.633 \\ -0.263 \end{bmatrix}, \tag{36}$$

where subscript (0) refers to step $k = 0$. It is important to scale the input to values close to a standard deviation of 1 with a mean of 0 for the training phase, so our input and desired output values were scaled up by a factor of 10. Applying the backpropagation technique with 10,000 iterations and $\alpha = 0.2$ produced the final weights given by

$$W^{(1)}_{(10000)} = \begin{bmatrix} w^{(1)}_{01(10000)} & w^{(1)}_{02(10000)} \\ w^{(1)}_{11(10000)} & w^{(1)}_{12(10000)} \end{bmatrix} = \begin{bmatrix} -6.2421 & 6.2331 \\ -2.0701 & -2.0789 \end{bmatrix} \tag{37}$$

and

$$\boldsymbol{w}^{(2)}_{(10000)} = \begin{bmatrix} w^{(2)}_{0(10000)} \\ w^{(2)}_{1(10000)} \\ w^{(2)}_{2(10000)} \end{bmatrix} = \begin{bmatrix} 1.9948 \\ -2.0308 \\ -1.9948 \end{bmatrix}. \tag{38}$$

The number of iterations and learning rate were chosen by looking at the least-squares error convergence of the algorithm and balancing the speed of convergence with any instability in the error plot. A small learning rate with many iterations is stable but slow to converge, whereas a large learning rate with fewer iterations is faster to converge but can be unstable.

The weights computed as a function of iteration number are shown in Figures 8a and 8b, where Figure 8a shows the four first-layer weights, and Figure 8b shows the three second-layer weights. The least-squares error as a function of iteration number is shown in Figure 8c, where the error was computed as follows for the k^{th} iteration:

$$E_{(k)} = \frac{1}{2} \sum_{i=1}^{4} (r_i - \hat{r}^{(2)}_{i(k)})^2 . \tag{39}$$

Both the weights and the least-squares error show an abrupt change in their behavior after iteration 2000. The error in Figure 8c can be divided into four separate regions. From iteration 1 to iteration 10, there is a dramatic drop in the error. Then, between iterations 10 and 2000, the change in the error becomes almost flat. Between iterations 2000 and 3000, there is another sharp decrease in the error, although not as dramatic as in the first 10 iterations. After iteration 3000, there is a gradual decline in the error toward zero.

On the error convergence plot, iteration 2000 is the value at the inflection point just before the algorithm starts to descend the second "hill." Why this is happening is hard to understand mathematically, but by looking at Figure 8c it appears that the neural network has "locked" into a local minimum between iterations 10 and 2000 and then suddenly "jumped" out of this local minimum after iteration 2000. A local minimum occurs

when the algorithm converges to what appears to be a stable solution, but the error is still quite large. We can look at the solution at 2000 iterations by stopping the neural network at this point in the calculation. The solution after 2000 iterations is extremely close to the least-squares solution, as shown by the least-squares error that has been plotted as the black circle on Figure 8c. This suggests that the least-squares solution is a very strong local minimum for the neural network.

The training of the neural network raised two important questions: how do we know that we are at a local minimum and not a global minimum, and how do we "escape" from it when we are trapped in it. The answer to the first question is that the error is usually quite large at a local minimum. But this is not always the case, and often a local minimum is difficult to detect. The answer to the second question is that it is very difficult to escape from a local minimum and that much research has been done in this area. For example, the technique of simulated annealing is often used to escape from local minima (see Masters, 1995).

At iteration 10,000, the error in Figure 8c has converged almost to zero, suggesting that a good solution has been derived. This solution (after scaling down by a factor of 10) is equal to

$$\hat{\boldsymbol{r}}^{(2)} = \begin{bmatrix} -0.0030 \\ +0.0999 \\ -0.0999 \\ +0.0030 \end{bmatrix}. \tag{40}$$

The values given in equation 40 are very close to the correct answer for the reflectivity, and, if we let the algorithm run for more iterations, we can arrive as close as we want to the exact answer. By using the weights in equations 37 and 38, we can confirm the values in equation 40 using the analytical neural network solution for the reflectivity as a function of the seismic data, using the following equation:

$$\hat{r}_i^{(2)} = \left[1.9948 - \frac{2.0308}{1+\exp\left(6.2421+2.0701 s_i\right)} - \frac{1.9948}{1+\exp\left(-6.2331+2.0789 s_i\right)} \right] / 10 . \tag{41}$$

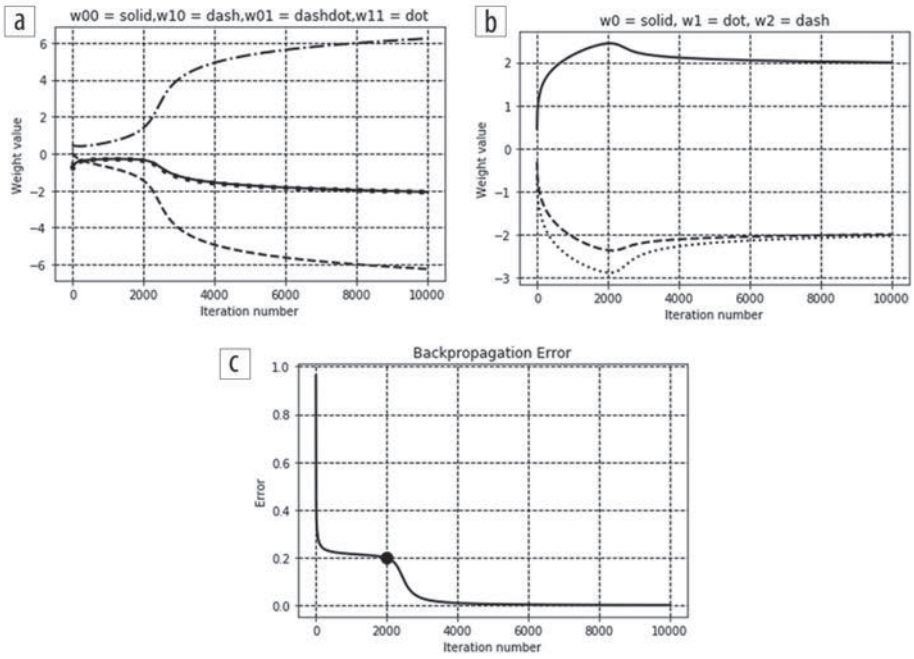

Figure 8. The convergence of the neural network weights and change in total error, where (a) shows the weights in the first layer, (b) shows the weights in the second layer, and (c) shows the total backpropagation error, all as a function of iteration number (the black point shows the error for least-squares regression).

Equation 41 shows that the output of a neural network is simply a mathematical transform of the input seismic values into the output reflection coefficients.

Another way to look at the solution in equation 41 is to crossplot the predicted reflectivity versus input seismic values for a range of values, as we did for linear regression. This is shown in Figure 9, which should be compared with the linear regression fit in Figure 4. The four known points in Figure 9 now show an almost perfect fit, but the curve is nonlinear. For seismic values above 0.3 and below –0.3, the curve starts to flatten off instead of staying straight as in Figure 4.

Figure 10 shows the estimated reflectivity after each of the techniques discussed in the preceding sections. Figure 10a shows both the true reflectivity and the result of deconvolution. If this perfect solution was achieved, the true impedance could be derived. Figure 10b shows the seismic response after wavelet tuning, which gives a very poor estimate of impedance. Figure 10c shows the least-squares scaling of Figure 10b, using our knowledge of the seismic trace (Figure 10b) and reflectivity (Figure 10a). Finally, Figure 10d shows the neural network estimate of reflectivity, where it should be noted that the first and last side lobes can be made as small as we want by increasing the number of iterations.

Conclusions

One conclusion to this study is that applying physics to a problem is better than applying a neural network because then the solution has a real physical meaning and is not just a mathematical transform. But actual geophysical problems are not that simple. When the geophysics is fully understood and applicable, it is always the better option. However, our geophysical solution is usually overly simplistic. (For example, in the real earth we have to consider dispersion of the wavelet, inhomogeneity, and anisotropy in the earth layers, etc.) Therefore, a neural network might find nonlinearities in the solution that we were unaware of in our theory. Second, our example consisted of very

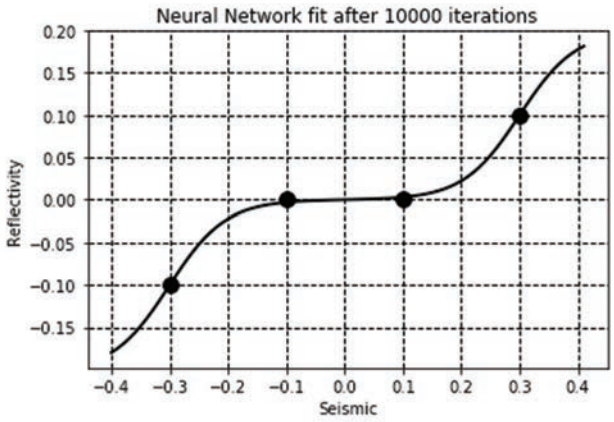

Figure 9. A plot of the predicted reflectivity versus seismic amplitude for the neural network.

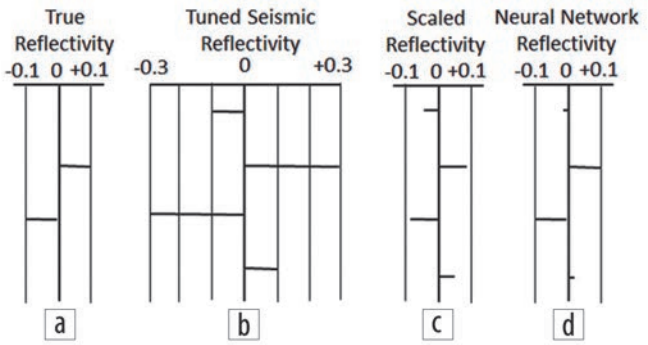

Figure 10. A summary of our various approaches to estimating the reflectivity from the seismic, where (a) shows the correct reflectivity and perfect deconvolution result, (b) shows the tuned reflectivity on the seismic trace, (c) shows the scaled reflectivity from least-squares regression (and the local minimum of the neural network), and (d) shows the neural network result.

few points. In real geophysical studies, we have large amounts of data, so a neural network might find hidden regularities in the data that we have overlooked. Furthermore, large amounts of data allow us to cross-validate our results by leaving parts of the data out of the initial training and using our trained weights to predict those parts of the data that were unknown to the neural network algorithm (Hampson et al., 2001).

The answer is therefore a judicious combination of both physical theories and machine learning techniques in our attempts to understand the earth. Above all, it is important to understand the theory behind both geophysics-based solutions and machine learning-based solutions. That was the key goal of this article. TLE

Acknowledgments

I want to thank my colleagues Dan Hampson and Jon Downton for our almost daily discussions about machine learning and Enders Robinson and Sven Treitel for their early papers on deconvolution, which at the start of my career showed me that the best way to understand complex theory is to use very clear examples.

Data and materials availability

Data associated with this research are available and can be obtained by contacting the corresponding author.

Corresponding author: brian.russell@cgg.com

References

Araya-Polo, M., J. Jennings, A. Adler, and T. Dahlke, 2018, Deep-learning tomography: The Leading Edge, **37**, no. 1, 58–66, https://doi.org/10.1190/tle37010058.1.

Claerbout, J. F., 1976, Fundamentals of geophysical data processing: McGraw-Hill Inc.

Hagan, M. T., H. B. Demuth, and M. Beale, 1996, Neural network design: PWS Publishing Company.

Hampson, D., J. S. Schuelke, and J. A. Quirein, 2001, Use of multiattribute transforms to predict log properties from seismic data: Geophysics, **66**, no. 1, 220–231, https://doi.org/10.1190/1.1444899.

Kim, Y., and N. Nakata, 2018, Geophysical inversion versus machine learning in inverse problems: The Leading Edge, **37**, no. 12, 894–901, https://doi.org/10.1190/tle37120894.1.

Lu, P., M. Morris, S. Brazell, and Y. Xiao, 2018, Using generative adversarial networks to improve deep-learning fault interpretation: The Leading Edge, **37**, no. 8, 578–583, https://doi.org/10.1190/tle37080578.1.

Masters, T., 1995, Advanced algorithms for neural networks: John Wiley & Sons Inc.

Naeini, E. Z., and K. Prindle, 2018, Machine learning and learning from machines: The Leading Edge, **37**, 886–891, https://doi.org/10.1190/tle37120886.1.

Rummelhart, D. E., G. E. Hinton, and R. J. Williams, 1986, Learning representations by back-propagating errors: Nature, **323**, 533–536, https://doi.org/10.1038/323533a0.

Schleicher, K., 2018, The conjugate gradient method: The Leading Edge, **37**, no. 4, 296–298, https://doi.org/10.1190/tle37040296.1.

Shewchuck, J., 1994, An introduction to the conjugate gradient method without the agonizing pain: School of Computer Science, Carnegie Mellon University, http://www-2.cs.cmu.edu/~jrs/jrs-papers.html, accessed 12 June 2019.

Widess, M. B., 1973, How thin is a thin bed?: Geophysics, **38**, no. 6, 1176–1180, https://doi.org/10.1190/1.1440403.

Widrow, B., and M. E. Hoff, 1960, Adaptive switching circuits: IRE WESCON Convention Record, Part 4, 96–104.