Geophysical Journal International

Geophys. J. Int. (2012) **189,** 1183–1202 doi: 10.1111/j.1365-246X.2012.05429.x

Data space reduction, quality assessment and searching of seismograms: autoencoder networks for waveform data

Andrew P. Valentine and Jeannot Trampert

Department of Earth Sciences, Universiteit Utrecht, P.O. Box 80021, 3508 *TA Utrecht, The Netherlands. E-mail: andrew@geo.uu.nl*

Accepted 2012 February 22. Received 2012 February 6; in original form 2011 August 9

SUMMARY

What makes a seismogram look like a seismogram? Seismic data sets generally contain waveforms sharing some set of visual characteristics and features—indeed, seismologists routinely exploit this when performing quality control 'by hand'. Understanding and harnessing these characteristics offers the prospect of a deeper understanding of seismic waveforms, and opens up many potential new techniques for processing and working with data. In addition, the fact that certain features are shared between waveforms suggests that it may be possible to transform the data away from the time domain, and represent the same information using fewer parameters. If so, this would be a significant step towards making fully non-linear tomographic inversions computationally tractable.

Hinton & Salakhutdinov showed that a particular class of neural network, termed 'autoencoder networks', may be used to find lower-dimensional encodings of complex binary data sets. Here, we adapt their work to the continuous case to allow the use of autoencoders for seismic waveforms, and offer a demonstration in which we compress 512-point waveforms to 32-element encodings. We also demonstrate that the mapping from data to encoding space, and its inverse, are well behaved, as required for many applications. Finally, we sketch a number of potential applications of the technique, which we hope will be of practical interest across all seismological disciplines, and beyond.

Key words: Time-series analysis; Neural networks, fuzzy logic; Self-organization; Statistical seismology; Wave propagation.

1 INTRODUCTION

Fig. 1 shows several different time-series. All are scaled to have amplitudes spanning the range [−1, 1], and for each, 500 samples are shown. Yet even a passing familiarity with seismic data is sufficient to identify which trace is a seismogram. Why?

The answer, of course, is that we expect a seismic signal to have certain characteristics and properties. Some of these are fairly general: all seismograms exhibit oscillatory behaviour on some timescale, and contain distinct wave packets. Others are specific to a particular class of data—for example, surface wave trains tend to have a particular appearance in a given frequency band. With exposure to seismic data, in undergraduate courses and beyond, we learn to recognize these features, and gain a feel for what a seismogram 'should' look like.

Mathematically, what does this mean? An N-point digital seismogram can be considered as a single entity: one point in an N-dimensional 'time-series space'. Our assertion that not all time-series are plausible seismograms implies that 'seismogram space' is a subset of this—it is possible to find a lower-dimensional space in which all N-point seismograms are still representable. Time-series clearly not of seismic origin would not be representable in this space.

Understanding this lower-dimensional space could provide many advantages and potential applications (Cottrell 2006). Clearly, it may help us understand and interpret the various features observed in seismic data sets, and may promote the development of new techniques for enhancing sensitivity or resolution in images of earth structure or source processes. Computation may be conducted in the lower-dimensional space, with attendant benefits in computation time—in particular, a reduction in data set dimension is likely to be of great importance if non-linear tomographic inversion is to be seriously pursued. Furthermore, knowledge of a space in which only plausible seismic data is representable opens up new avenues for the automated processing of real data.

The representation of data sets using fewer parameters has obvious links to data compression, and in this context two papers by Shannon (1948, 1949) may be of particular interest, laying out mathematical foundations for transmission of data through noisy channels. However, it should be remembered that this is a different situation from that encountered by seismologists, for whom noise is already present in the data set.

In recent years, compressive sensing (e.g. Donoho 2006; Tsaig & Donoho 2006; Candès & Wakin 2008) has attracted much attention. This is targeted at data-acquisition problems, for signals which are known to have a sparse representation in some basis.

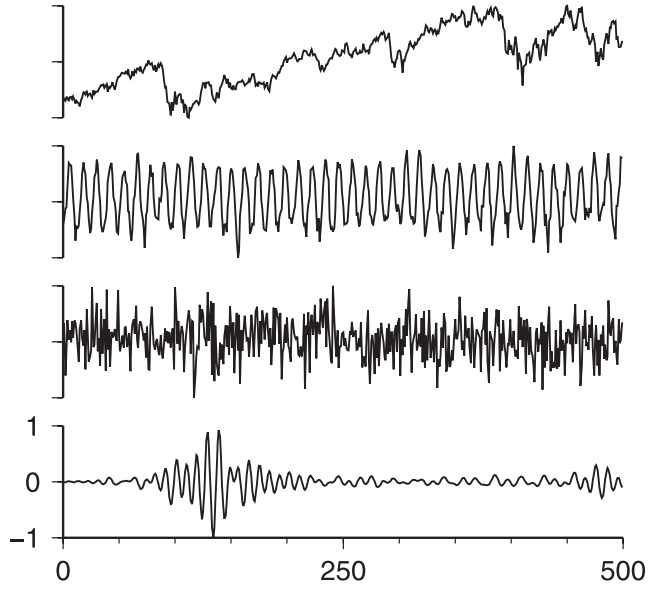

Figure 1. 'Spot the seismogram'. Four 500-point time-series, normalized to take amplitudes in the range $[-1, 1]$—but the seismogram has sufficient characteristic features to make it instantly recognizable. From top: FTSE 100 closing prices, 2009 June–2011 May; monthly mean temperature for central England, 1950–1991 (Parker *et al.* 1992); Gaussian random noise; long-period surface wave seismogram.

Rather than recording individual samples in space or in time, it is possible to measure particular linear functionals of these samples, from which the original signal can be reconstructed. This has found application in geophysics, particularly in exploration seismology, where similar methods were already known: see, for example, Herrmann *et al.* (2009) or Wang *et al.* (2011). Also of interest are wavelet-based methods, which have found a range of different applications in seismology (e.g. Chakraborty & Okaya 1995; Operto *et al.* 2002; Simons *et al.* 2011). Typically, these involve techniques for developing and exploiting a sparse basis relevant to a particular application.

We stress, at the outset, that the approaches discussed in this paper are not intended for the compression of data sets with a view to reducing storage or transmission costs. Such tasks are best handled by one of the mature compression algorithms that are readily available (e.g. Ziv & Lempel 1977, implemented as `gzip`). These typically exploit repetitive sequences in the binary representation of data, and are fast, lossless and widespread. Our approach attempts to exploit patterns in the data set itself, and is typically lossy; however, it attempts to isolate the characteristics specific to a particular seismogram amidst those common to *all* seismograms (or all seismograms of a particular class). These characteristics may then be the subject of further computation or analysis, with the dimension reduction or 'compression' thus achieved leading to greater computational efficiency. Our method also allows for the identification of waveforms that do *not* share the general characteristics of a particular class of data, providing an efficient approach for automating the extraction of subsets from large waveform catalogues.

Our approach is based on the work of Hinton & Salakhutdinov (2006), and employs a particular class of neural network—the 'autoencoder'. We begin by setting out the problem in general terms: assuming that seismograms *can* be represented in a lower-dimensional space, what properties would the transformation have and how might it be used? We then introduce autoencoder networks as a route to identifying and implementing this transformation, and demonstrate that they do indeed allow seismic data sets to be represented with fewer parameters. Finally, we show one potential application—quality assessment of seismograms, as in Valentine & Woodhouse (2010)—and discuss other promising avenues.

2 ENCODINGS AND DECODINGS: SOME GENERAL PROPERTIES

To motivate the current work, and to provide some general insight, we first consider the abstract problem of mapping a high-dimensional data set into a lower-dimensional space; in doing so, we try to strike a balance between mathematical formalism, and clarity. We suppose that some *class* of data naturally exists in N dimensions: for our purposes, this can be taken to be N-point digital seismograms. For most practical purposes, we further assume that the 'class' of seismogram is restricted such that its members would be recognized by a seismologist as 'similar'—*inter alia*, we expect them to share a common sampling rate, and be processed in such a way that the traces are comparable. For some applications, it may be desirable to restrict the class further: for example, it might contain all continuous N-point sequences recorded by a particular station in some time frame, or be the set of all N-point seismograms recorded worldwide starting from a given time. However, the class is defined, we denote the set of all possible members by $\mathbb{S}^N \subset \mathbb{R}^N$, where $\mathbb{R}^N$ has its common meaning as the N-dimensional space of real numbers.

2.1 Encoding and decoding operations

We intend here that $\mathbb{S}^N$ is the set of theoretical or 'true' data, and may be infinite; typically, we have access to a finite number of noisy samples from $\mathbb{S}^N$. We now assume that for any example $\mathbf{s}_i$ drawn from $\mathbb{S}^N$, it is possible to compute an M-element 'encoding'—that is, we can transform $\mathbf{s}_i$ into an M-dimensional space. We denote this encoding by $\mathbf{t}_i$, and represent the transformation by

$$\mathbf{t}_i = \underset{N\to M}{\mathrm{enc}}\,(\mathbf{s}_i)\,. \tag{1}$$

For present purposes, we simply assume that such an encoding operation exists; we also restrict ourselves to the case where $M < N$. We define $\mathbb{T}^M$ to be the set of all possible $\mathbf{t}_i$ generated from members of $\mathbb{S}^N$—in mathematical terms, it is the image of $\mathbb{S}^N$ under the encoding operation, denoted

$$\mathbb{T}^M = \underset{N\to M}{\mathrm{enc}}\left[\mathbb{S}^N\right]. \tag{2}$$

For each element in $\mathbb{T}^M$, we also suppose that there is a 'decoding' operation, so that

$$\mathbf{s}'_i = \underset{M\to N}{\mathrm{dec}}\,(\mathbf{t_i})\,, \tag{3}$$

and we define

$$\mathbb{S}'^N = \underset{M\to N}{\mathrm{dec}}\left[\mathbb{T}^M\right]. \tag{4}$$

Thus, the combined effect of encoding and decoding operations is to map each element of $\mathbb{S}^N$ onto an element of $\mathbb{S}'^N$.

In the ideal case, $\mathrm{dec}(\mathbf{t}) = \mathrm{enc}^{-1}(\mathbf{t})$, so that $\mathbb{S}'^N = \mathbb{S}^N$ and $\mathbf{s}'_i = \mathbf{s}_i$. However, in general this may not hold, and we quantify the 'reconstruction error' for a particular example by some metric, such as the sum of squares difference between original and decoded seismogram

$$E_i = \frac{1}{2}\left(\mathbf{s}'_i - \mathbf{s}_i\right)\cdot\left(\mathbf{s}'_i - \mathbf{s}_i\right). \tag{5}$$

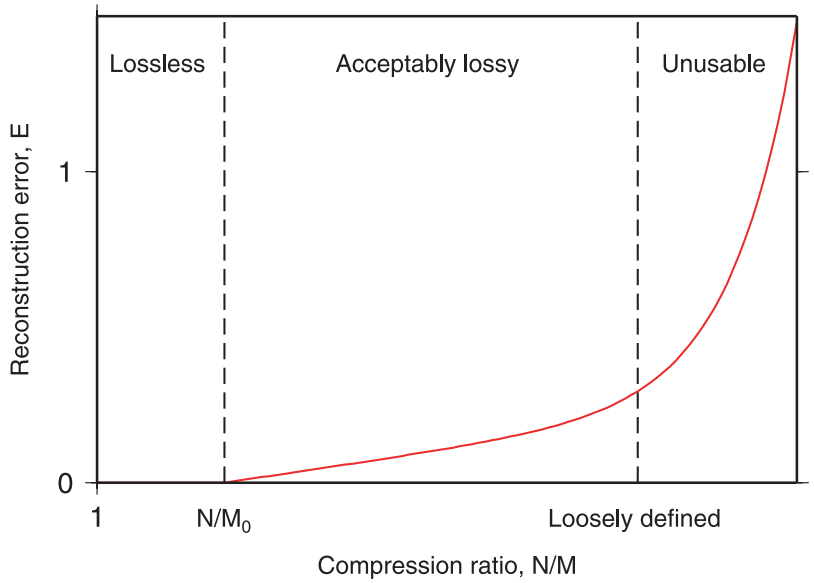

Figure 2. The cost of compression. Sketch of relationship between compression ratio N/M and reconstruction error, $\bar{E}$. When attempting to represent an N-dimensional system $\mathbb{S}^N$ in M dimensions ($M < N$), we expect to move through three regimes as M is reduced: 'lossless', where all elements of $\mathbb{S}^N$ can be perfectly recovered; 'acceptably lossy', where most features of $\mathbb{S}^N$ can be reproduced, and only minor differences between the original and its reconstruction are observed and 'unusable', where the compression rate is too high to allow even the basic structure of $\mathbb{S}^N$ to be represented. The width of the first two regimes will vary between data sets, and either or both may not be observed in a particular case.

Clearly, $E_i = 0$ for a perfect reconstruction. The average error across all possible members of $\mathbb{S}^N$ is then given by

$$\bar{E} = \lim_{Q\to\infty} \frac{1}{Q} \sum_{i=1}^{Q} E_i \,, \tag{6}$$

which can be estimated using a finite number of representative samples from the set, because the variability of samples from $\mathbb{S}^N$ is taken to be relatively small. For a given $\mathbb{S}^N$ and encoding dimension M, an optimal encoder–decoder pair is one which minimizes $\bar{E}$.

Intuitively, we expect the best value of $\bar{E}$ attainable to depend upon the ratio N/M, with three regimes as shown in Fig. 2. Some lossless compression may be possible, depending on the properties of $\mathbb{S}^N$, and we define M_0 to be the dimension of the smallest space in which $\mathbb{S}^N$ can be represented without loss. If we attempt to achieve a compression ratio greater than N/M_0, the encoding–decoding operation will be lossy, but for many applications a small increase in $\bar{E}$ may be acceptable for lower M. Eventually, however, M will become too small to allow even the basic structure of $\mathbb{S}^N$ to be represented, and $\bar{E}$ will grow rapidly. In this paper, we will be concerned mainly with the 'acceptably lossy' regime.

2.2 The encoding space, $\mathbb{T}^M$

Any sensible choice of encoder and decoder will ensure that every possible output from the encoding operation is a valid input to the decoding operation; that is

$$\mathcal{R}\left(\underset{N\to M}{\text{enc}}\right) \subseteq \mathcal{D}\left(\underset{M\to N}{\text{dec}}\right) , \tag{7}$$

where $\mathcal{D}$ and $\mathcal{R}$ are used to denote the domain and range (image) of a function, respectively. From the definition in eq. (2), we must have

$$\mathbb{T}^M \subseteq \mathcal{R}\left(\underset{N\to M}{\text{enc}}\right) , \tag{8}$$

with equality for an ideal encoder–decoder system in any case where $M < M_0$: in other words, an ideal encoder will make use of its full range of output values for representing $\mathbb{S}^N$. A consequence of this is that any point within $\mathcal{D}(\text{enc})$ will encode to a point in $\mathbb{T}^M$, regardless of whether it lies in $\mathbb{S}^N$.

An ideal encoder–decoder system seeks to identify the characteristics common to all members of $\mathbb{S}^N$, and encapsulates these within the encoding and decoding functions. The information contained in each $\mathbf{t}_i \in \mathbb{T}^M$ is, therefore, only that necessary to distinguish between individual samples within the set. If the encoding is lossy, and $\bar{E} > 0$, some distinct members of $\mathbb{S}^N$ must map to the same point in $\mathbb{T}^M$: essentially, the finer details of $\mathbb{S}^N$ become blurred. In general, if $\bar{E}$ is to be minimal, only elements of $\mathbb{S}^N$ that are 'close' may share an encoding in $\mathbb{T}^M$, where the precise definition of 'close' depends on the detail of $\bar{E}$ and its definition. In general, we wish the mapping to be 'well behaved', and preserve relationships between elements: if two examples are 'close' in $\mathbb{S}^N$, we desire them to be 'close' in $\mathbb{T}^M$. This allows many calculations that are typically performed on the data to be translated into the encoding space, with an accuracy governed by $\bar{E}$. However, it is not possible to state general results without knowledge of the form of the encoding operation.

2.3 The data space, $\mathbb{S}^N$: a test for membership

We have yet to define the domain of the encoding operation—so far, we have simply required that it exists for every element in $\mathbb{S}^N$. Formally, this implies

$$\mathbb{S}^N \subseteq \mathcal{D}\left(\underset{N\to M}{\text{enc}}\right) , \tag{9}$$

but the reader will recall our statement at the start of this section that $\mathbb{S}^N$ represents the theoretical data space, and its elements are free of noise and other difficulties. A practical encoder should be able to handle real data; in any case, it is likely to be difficult to formulate an *a priori* definition of $\mathbb{S}^N$. We therefore require that

$$\mathcal{D}\left(\underset{N\to M}{\text{enc}}\right) = \mathbb{R}^N . \tag{10}$$

What happens if we encode some N-element vector $\mathbf{r}$ that is not a member of $\mathbb{S}^N$? From eq. (10), $\mathbf{r}$ will be a valid input to the encoding operation; as discussed following eq. (8), the output will lie in $\mathbb{T}^M$. If we then decode the result, eq. (4) implies that we obtain a reconstruction, $\mathbf{r}'$, lying within $\mathbb{S}'^N$. Thus, the reconstruction of a signal that *is* in $\mathbb{S}^N$ is indistinguishable from the reconstruction of one that is not. However, the reconstruction errors (eq. 5) will differ considerably: members of $\mathbb{S}^N$ will have low values, broadly comparable with $\bar{E}$; non-members will typically show large errors. This provides a straightforward test for whether a given trace is likely to be a sample from $\mathbb{S}^N$, to within a tolerance governed by $\bar{E}$.

2.4 From theory to practice

Thus far, we have simply assumed that the encoding and decoding mappings exist; we have offered no suggestion as to how they might be realized in practice. It might appear difficult to do so, particularly given the difficulty in formulating a robust and comprehensive description of $\mathbb{S}^N$. However, neural networks provide a possible solution, and in particular, a class of networks known as 'autoencoders' (Hinton & Salakhutdinov 2006). The power of neural networks lies in their ability to 'learn' a mapping, given only samples of its inputs and outputs. Although we may not be able to fully define $\mathbb{S}^N$ for any given situation, we can easily obtain a large number of samples from it. As we shall see, this allows us to discover encoding and decoding operations that satisfy the various requirements of eqs (1)–(10).

To conclude this section, we note that although we cannot define $\mathbb{S}^N$ *a priori*, we can map out $\mathbb{S}'^N$ once we have obtained an optimal encoder–decoder pair. From eq. (4), $\mathbb{S}'^N$ is the image of $\mathbb{T}^M$ under the decoding operation, and we have already observed that $\mathbb{T}^M$ should be equivalent to $\mathcal{R}$ (enc), which will be known. A uniform sampling of $\mathbb{T}^M$ can then be used to assess the form of $\mathbb{S}'^N$. Provided $\bar{E}$ is small, this provides a good estimate of $\mathbb{S}^N$, although the topology of $\mathbb{S}'^N$ will be influenced by the form of the encoding and decoding mappings.

3 LEARNING MAPPINGS BETWEEN SPACES: NEURAL NETWORKS

Neural networks are thought to provide a mathematical analogue for the way in which humans learn to process and interpret information. Their key advantage lies in their ability to model systems that are not well understood: unlike classical approaches, there is no need for the user to develop a detailed understanding of the system dynamics before implementing a model. Instead, the network is provided with a set of inputs, and the corresponding desired outputs; from these, it attempts to 'learn' the underlying relationship. After 'training', the network may be used for prediction, much like a traditional model.

A network consists of a large number of interconnected 'neurons'—typically simple functions that take many inputs and return a single output, with the precise relationship governed by a number of adjustable weights. Generally, neurons are arranged in 'layers', with the outputs from one layer being used as inputs to the next. By varying the weights, the behaviour of the network can be altered, and 'training' involves optimizing the weights for a particular case. It can be shown that arbitrarily complex mathematical functions are representable by neural networks, although factors such as size and architecture may limit the range of any particular network.

Neural computing is an active research field, and for a thorough introduction the interested reader is encouraged to consult one of the many books on the subject (e.g. Bishop 1995; Mackay 2003). A number of introductory surveys may also be found, written from a variety of perspectives: examples include those by Kohonen (1988), Cheng & Titterington (1994) and Basheer & Hajmeer (2000). Reviews aimed at researchers in the geosciences include those of van der Baan & Jutten (2000) and Mas & Flores (2008), and geophysical applications are covered in a book by Sandham & Leggett (2004). Such applications include non-linear inversion (e.g. Meier *et al.* 2007; Ho 2009), feature classification (e.g. Shimshoni & Intrator 1998; Tingdahl & de Rooij 2005) and data selection (e.g. Dai & MacBeth 1995; Gentili & Micheli 2006; Valentine & Woodhouse 2010; Diersen *et al.* 2011).

3.1 Autoencoder networks

In this paper, we will make use of a particular class of neural network, known as an 'autoencoder'. These were set out by Hinton & Salakhutdinov (2006), and are, essentially, networks trained to output faithful reproductions of their inputs. However, the network architecture is such that the intermediate layers contain fewer neurons than the top- and bottom-level layers (see Fig. 3). The network can therefore be regarded as a connected encoder–decoder pair, with the layer containing the smallest number of neurons providing the encoded representation of the inputs. Because each layer depends only on the values of the previous layer, knowledge of the encoding is sufficient to compute the corresponding reconstruction. Hinton & Salakhutdinov (2006) focus on autoencoders for binary data sets; it will be necessary to adapt this to the continuous case.

It is common to describe autoencoders by specifying the number of nodes in each layer; thus, that shown in Fig. 3 is a 7–6–4–6–7 autoencoder. For reasons that shall later become apparent, we shall

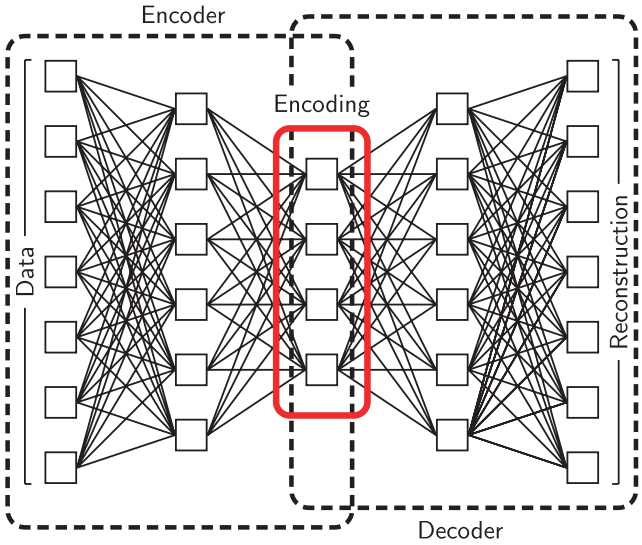

Figure 3. Architecture of an autoencoder network. Successive layers of nodes detect patterns in input data, and use these to generate an encoded representation of the data. The corresponding decoder takes encodings, and attempts to reconstruct the original inputs. The network training algorithm involves adjusting the behaviour of each node to bring reconstructions closer to data. Networks may be described by the number of nodes in each layer; shown is a 7–6–4–6–7 autoencoder. In practice, several more layers would be used.

restrict ourselves to what might be termed 'symmetric' networks, where the layers in the decoder mirror those in the encoder. We note here a potential source of confusion: neural networks tend to be described in terms of the number of layers of *neurons*, whereas it is arguably more natural to think of autoencoders in terms of the number of layers of *nodes*. For the avoidance of doubt, the network depicted in Fig. 3 has five layers of nodes, connected by four layers of neurons.

3.1.1 Network architecture

Autoencoders are distinguished from more general neural networks by the fact that their outputs are desired to be the same as their inputs. However, the manner in which the network operates, and may be trained, is quite standard—although as Hinton & Salakhutdinov (2006) have shown, the addition of a layer-by-layer pre-training stage allows much faster convergence. We shall return to this shortly; however, we first summarize the equations governing the network as a whole. Because neural networks are likely to be unfamiliar to many readers, a more detailed derivation of these results can be found as an Appendix to this paper.

In general, an individual neuron may have a number of inputs. These may be represented by a vector quantity, $\mathbf{x}$, of dimension l. Associated with the neuron is an l-element weight vector, denoted $\mathbf{w}$, controlling the significance attached to each input by that neuron. The neuron also has an internal 'bias', b, to control the threshold at which the neuron's output changes, and a 'sensitivity', a, governing the overall responsiveness of the neuron. All neurons used in this study output a single value, according to

$$y = f(ab + a(\mathbf{w}\cdot\mathbf{x}))\,, \tag{11}$$

where $f(x)$ represents some 'activation function'. In this paper, we restrict ourselves to the case where each neuron in the network implements a logistic function, with general form

$$f(x) = f_0 + \frac{f_1 - f_0}{1+\exp(-x)}\,, \tag{12}$$

where f_0 and f_1 govern the lower and upper limits of the output. It has derivative

$$\begin{aligned}\frac{\mathrm{d}f}{\mathrm{d}x} &= (f_1 - f_0)\frac{\exp(-x)}{[1+\exp(-x)]^2}\\ &= \frac{1}{f_1 - f_0}\,[f(x) - f_0]\,[f_1 - f(x)]\,,\end{aligned} \tag{13}$$

and is shown in Fig. 4.

As previously mentioned, a network consists of multiple layers of neurons, with all neurons in one layer receiving the same inputs. However, the weights and biases are initially randomized, so that the *output* of one neuron differs from that of the next, and so that they may exhibit sensitivity to different aspects of the inputs. We shall denote the outputs of the nth layer of neurons by the vector quantity $\mathbf{x}^{(n)}$, and this is used as the input to the $(n+1)$th layer. We therefore denote the inputs to the first layer—and thus to the network as a whole—by $\mathbf{x}^{(0)}$. The weights connecting two adjacent layers are denoted by the matrix $\mathbf{W}^{(n)}$, and the sensitivities and biases for all neurons in a given layer can be denoted by the vector quantities $\mathbf{a}$ and $\mathbf{b}$, respectively. Each layer of the network then implements

$$\mathbf{x}^{(n)} = \mathbf{f}\left(\mathbf{a}^{(n)} * \mathbf{b}^{(n)} + \mathbf{a}^{(n)} * \left(\mathbf{W}^{(n)}\mathbf{x}^{(n-1)}\right)\right)\,, \tag{14}$$

where we use '$*$' to denote elementwise multiplication of vector or

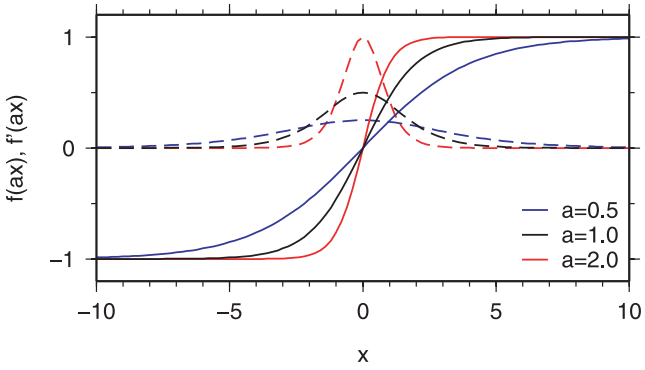

Figure 4. The logistic function. Plots of $f(ax) = f_0 + \frac{f_1 - f_0}{1+\exp(-ax)}$ (solid lines) and their derivatives, $\frac{\partial f}{\partial x}$ (dashed lines), for three values of a, and with $f_0 = -1, f_1 = 1$. Note that the logistic function approaches a step function as a increases.

matrix quantities, so that

$$\begin{aligned}[\mathbf{a} * \mathbf{b}]_i &= a_i b_i\\ [\mathbf{A} * \mathbf{b}]_{ij} &= A_{ij} b_i\\ [\mathbf{A} * \mathbf{B}]_{ij} &= A_{ij} B_{ij}\end{aligned} \tag{15}$$

for vector quantities $\mathbf{a}$ and $\mathbf{b}$, and matrix quantities $\mathbf{A}$ and $\mathbf{B}$. We use $\mathbf{f}(\mathbf{x})$ to denote a vectorized form of $f(x)$, so that

$$[\mathbf{f}(\mathbf{x})]_i = f(x_i)\,. \tag{16}$$

If the nth layer contains $K^{(n)}$ neurons, it will be apparent that $\mathbf{a}^{(n)}$ and $\mathbf{b}^{(n)}$ must have $K^{(n)}$ elements, and that $\mathbf{W}^{(n)}$ has dimension $K^{(n)} \times K^{(n-1)}$. Each row of $\mathbf{W}$ corresponds to $\mathbf{w}$ in eq. (11) for one neuron. Because we typically wish to work with data sets containing numerous examples, we note that eq. (14) can describe operations on the entire data set simultaneously if the various $\mathbf{x}^{(n)}$ are regarded as $K^{(n)} \times Q$ matrices, where Q represents the number of examples in the data set.

We define L such that $\mathbf{x}^{(L)}$ represents the output layer of nodes—the 'reconstruction' of Fig. 3. Because we are indexing the input layer as $\mathbf{x}^{(0)}$, the 'encoding' is represented by $\mathbf{x}^{(L/2)}$; to translate to the notation used in Section 2, $\mathbf{x}^{(0)} \to \{\mathbf{s}_i\} \subset \mathbb{S}^N$, $\mathbf{x}^{(L/2)} \to \{\mathbf{t}_i\} \subset \mathbb{T}^M$ and $\mathbf{x}^{(L)} \to \{\mathbf{s}'_i\} \subset \mathbb{S}'^N$. Clearly, $K^{(0)} = K^{(L)} = N$, and $K^{(L/2)} = M$. Thus, eq. (14) fully describes the operation of our network.

3.1.2 Network training

In order to train the network, we must define an error function, which we shall seek to minimize. We wish the network to produce outputs as close as possible to its inputs: for a perfect encoder–decoder pair, they should be identical, as set out in Section 2. As suggested in eq. (5), we therefore choose to minimize the average Euclidean distance between desired and actual outputs across the entire data set

$$E = \frac{1}{2Q}\sum_{i,j}\left(x_{ij}^{(L)} - x_{ij}^{(0)}\right)^2\,. \tag{17}$$

This can be achieved using, for example, the 'back-propagation algorithm', a standard method in the neural network literature, which essentially adopts a gradient descent approach. We define the quantity $\mathbf{\Delta}^{(n)}$, representing the back-propagated error in layer n of the network

$$\mathbf{\Delta}^{(n-1)} = \mathbf{W}^{(n)\mathrm{T}}\left(\mathbf{\Delta}^{(n)} * \mathbf{u}^{(n)} * \mathbf{a}^{(n)}\right)\,, \tag{18}$$

$$\mathbf{\Delta}^{(L)} = \mathbf{x}^{(L)} - \mathbf{x}^{(0)} . \tag{19}$$

We also define the matrix quantity $\mathbf{u}^{(n)}$, based on the derivative of the logistic function given in eq. (13), as

$$u_{ij}^{(n)} = \frac{1}{f_1 - f_0} \left(x_{ij}^{(n)} - f_0 \right) \left(f_1 - x_{ij}^{(n)} \right) . \tag{20}$$

It can then be shown (as in the Appendix) that the overall network error is reduced by updating all biases, weights and sensitivities simultaneously, according to

$$b_i^{(n)} \rightarrow b_i^{(n)} - \frac{\eta}{Q} \sum_j \Delta_{ij}^{(n)} u_{ij}^{(n)} a_i^{(n)} \tag{21}$$

$$W_{ij}^{(n)} \rightarrow W_{ij}^{(n)} - \frac{\eta}{Q} \sum_k \Delta_{ik}^{(n)} a_i^{(n)} u_{ik}^{(n)} x_{jk}^{(n-1)} \tag{22}$$

$$a_i^{(n)} \rightarrow a_i^{(n)} - \frac{\eta}{Q} \sum_j \Delta_{ij}^{(n)} u_{ij}^{(n)} \left(b_i^{(n)} + \sum_k W_{ik}^{(n)} x_{kj}^{(n-1)} \right) . \tag{23}$$

Here, η is a positive constant that governs the 'learning rate' of the network. Provided that η is sufficiently small that second-order derivatives may be neglected, repeated application of these formulae over many iterations allows the network to find optimal values for all weights.

Typically, we wish the network to assimilate the *general* properties of the training set, rather than the precise form of each trace—as we shall see, the latter tends to have an adverse effect on the network's ability to handle examples that were not present in the training set. It is, therefore, usual to assess the evolution of the network during training through the use of a 'monitoring' data set. This is chosen to be equivalent to—but independent from—the data set used during training ($\mathbf{x}^{(\mathbf{0})}$): in the notation of Section 2, it contains further samples from $\mathbb{S}^N$. As training progresses, the reconstruction error for this set is monitored, and the training algorithm is terminated if this becomes essentially static over a number of iterations, or if it begins to increase.

3.1.3 Noise

In a similar vein, it may be desirable to introduce noise into the system during training, so that the 'precise form' of each trace becomes blurred. By introducing small random variations in the training data, we desensitize the network to minor details and emphasize the overall characteristics of each trace. In some sense, this is similar to the need to make use of regularization in geophysical inverse problems.

We achieve this by adding random, Gaussian noise to each point in the training set; we repeat this afresh on each iteration of the training algorithm, so that the training data received by the network is never identical between iterations. This therefore corresponds to the transformation

$$x_{ij}^{(0)} \rightarrow x_{ij}^{(0)} + \mathcal{G}(0, \sigma^{(A)}) , \tag{24}$$

where $\mathcal{G}(\mu, \sigma)$ denotes a random sample from a Gaussian distribution of mean μ and standard deviation σ.

3.2 Network pre-training: continuous restricted Boltzmann machines

In principle, given an appropriate data set, we are now in a position to train an autoencoder. We would start with $w_{ij}^{(n)}$ and $b_i^{(n)}$ randomized, typically to a Gaussian distribution (see below), set all $a_i^{(n)} = 1$, and use eqs (21)–(23) repeatedly. However, this turns out to be beset by problems. Initially, the outputs from the network are far removed from the inputs, and depending on the topology of the 'error surface' defined by eq. (17), our training algorithm may not arrive at the desired, global, minimum. Even if it does, convergence will typically be extremely slow. As a result, it is helpful to 'pre-train' the network, so that when we begin to apply eqs (21)–(23), we are in the vicinity of the desired minimum.

Hinton & Salakhutdinov (2006) suggest the use of 'restricted Boltzmann machines' (RBMs) for this. These are simple, two-layer networks, which employ a stochastic learning rule to identify 'features' within input data (Ackley *et al.* 1985; Hinton 2002; Mackay 2003; Hinton 2010). Generally, their application lies in the recognition of familiar patterns amidst noise. Conventional RBMs have binary inputs and outputs, but it is possible to extend this to the continuous case. We shall follow the treatment in Chen & Murray (2003).

3.2.1 Continuous restricted Boltzmann machines

The operation of a 'continuous restricted Boltzmann machine' (CRBM) is straightforward, and has clear similarities with the network architecture already discussed. The CRBM is a two-layer network; the value, x_i^{h} of the ith node in the second, 'hidden' layer is related to the values of the K_{v} 'visible' nodes x_j^{v} via

$$x_i^{\mathrm{h}} = f \left(a_i^{\mathrm{h}} \left[b_i^{\mathrm{h}} + \sum_{j=1}^{K_{\mathrm{v}}} w_{ij} x_j^{\mathrm{v}} + \mathcal{G} \left(0, \sigma^{(C)} \right) \right] \right) , \tag{25}$$

where $f(x)$ is the logistic function defined in eq. (12), and where $\mathcal{G}(\mu, \sigma)$ again represents a random sample from a Gaussian distribution. As before, we have weights w_{ij}, a bias b_i^{h} and a sensitivity, a_i^{h}. However, the network is simpler, and may operate in a stochastic fashion. As we shall see, it also has a quite different training rule.

Eq. (25) provides a rule for updating the hidden nodes, given values for the visible nodes. We can specify a similar rule to update the visible nodes,

$$x_j^{\mathrm{v}} = f \left(a_j^{\mathrm{v}} \left[b_j^{\mathrm{v}} + \sum_{i=1}^{K_{\mathrm{h}}} w_{ij} x_i^{\mathrm{h}} + \mathcal{G} \left(0, \sigma^{(C)} \right) \right] \right) . \tag{26}$$

Note that the visible-to-hidden and hidden-to-visible connections share the same (transposed) weight matrix, but in general have different biases and sensitivities.

CRBM training seeks to find and enhance correlations between the values of visible and hidden nodes. Typically, this is done via an algorithm known as 'minimizing contrastive divergence' (Hinton 2002). The visible nodes of the network are initially set to match some example from the training data set; we denote their values by the vector, $\mathbf{x}^{\mathrm{v}}$. Using eq. (25), we can compute states for the hidden nodes, $\mathbf{x}^{\mathrm{h}}$. We then use these in conjunction with eq. (26) to generate a 'reconstruction' of the inputs, $\hat{\mathbf{x}}^{\mathrm{v}}$; finally, this reconstruction is used to obtain updated hidden values, $\hat{\mathbf{x}}^{\mathrm{h}}$. We then update the weights, biases and sensitivities according to

$$b_i^{\mathrm{h,v}} \rightarrow b_i^{\mathrm{h,v}} + \eta \left[\left\langle x_i^{\mathrm{h,v}} \right\rangle - \left\langle \hat{x}_i^{\mathrm{h,v}} \right\rangle \right] \tag{27}$$

$$w_{ij} \rightarrow w_{ij} + \eta \left[\left\langle x_i^{\mathrm{h}} x_j^{\mathrm{v}} \right\rangle - \left\langle \hat{x}_i^{\mathrm{h}} \hat{x}_j^{\mathrm{v}} \right\rangle \right] \tag{28}$$

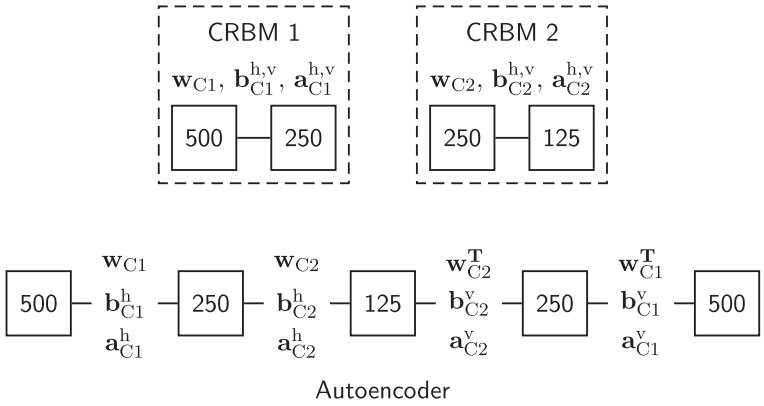

Figure 5. Assembling the autoencoder. A 500-250-125-250-500 autoencoder is generated using weights, biases and sensitivities taken from two trained CRBMs; one has 500 visible and 250 hidden nodes, the other has 250 visible and 125 hidden nodes. The second CRBM is trained using data that has been 'encoded' by the first. The CRBMs are used to initialize the autoencoder with reasonable values; these are then refined according to the network training algorithm. After Hinton & Salakhutdinov (2006).

$$a_i^{\mathrm{h,v}} \to a_i^{\mathrm{h,v}} + \frac{\eta}{\left(a_i^{\mathrm{h,v}}\right)^2}\left[\left\langle\left(x_i^{\mathrm{h,v}}\right)^2\right\rangle - \left\langle\left(\hat{x}_i^{\mathrm{h,v}}\right)^2\right\rangle\right], \tag{29}$$

where angled brackets $\langle\chi\rangle$ denote the average value of χ across all samples in the training set. Eq. (29), in particular, relies on an approximation to an integral; for further details, see Chen & Murray (2003).

This training process encourages the CRBM to find and enhance relationships between visible and hidden nodes, so that the hidden nodes become—in some sense—an encoding of the visible nodes. The stochastic dynamics of the system desensitize the network to noise, and encourage fuller exploration of the possible states of the system than is seen with gradient descent methods such as back-propagation. In addition, each CRBM contains a fraction of the number of free parameters in the autoencoder, leading to better convergence properties. As a result, the CRBM learning algorithm provides a useful technique to pre-train each layer of our autoencoder separately.

3.2.2 Application to pre-training

Suppose, for the sake of argument, that we wish to generate a 500-250-125-250-500 autoencoder; we begin by training a CRBM with 500 visible nodes, and 250 hidden nodes (see Fig. 5). Once a certain number of training iterations have been conducted, we use that CRBM to convert our training database, containing numerous 500-element vectors, into a new training set containing 250-element vectors. These are used to train a new CRBM, with 250 visible nodes and 125 hidden ones. We are then in a position to assemble the autoencoder, using the weights, biases and sensitivities from the CRBMs: we initialize the autoencoder by identifying

$$\begin{aligned}
&\mathbf{w}_{C1} \to \mathbf{W}^{(1)}; \quad \mathbf{b}^{h}_{C1} \to \mathbf{b}^{(1)}; \quad \mathbf{a}^{h}_{C1} \to \mathbf{a}^{(1)};\\
&\mathbf{w}_{C2} \to \mathbf{W}^{(2)}; \quad \mathbf{b}^{h}_{C2} \to \mathbf{b}^{(2)}; \quad \mathbf{a}^{h}_{C2} \to \mathbf{a}^{(2)};\\
&\mathbf{w}^{T}_{C2} \to \mathbf{W}^{(3)}; \quad \mathbf{b}^{v}_{C2} \to \mathbf{b}^{(3)}; \quad \mathbf{a}^{v}_{C2} \to \mathbf{a}^{(3)};\\
&\mathbf{w}^{T}_{C1} \to \mathbf{W}^{(4)}; \quad \mathbf{b}^{v}_{C1} \to \mathbf{b}^{(4)}; \quad \mathbf{a}^{v}_{C1} \to \mathbf{a}^{(4)}.
\end{aligned} \tag{30}$$

Here, subscript C1 refers to the first (500-250) CRBM, and C2 denotes the second (250-125). Unsubscripted quantities relate to the autoencoder. At this point, we proceed with training the autoencoder according to eqs (21)–(23); in particular, there is no requirement for the decoder-layer weights to retain the transpose relationship with the encoder-layer counterparts. We make no further use of the CRBMs themselves.

3.3 Practicalities

As always, a number of practical matters must be addressed in order to successfully implement this theory for seismic data sets. We note that much helpful advice on CRBMs and their training may be found in Hinton (2010), and that this has influenced our approach.

3.3.1 Data preparation and weighting

Implicit in the use of logistic neurons of the form of eq. (12) is that only values in the range $[f_0, f_1]$ are representable. We, therefore, scale all waveforms by their maximum amplitude, to ensure that all data points are in the range $[-1, 1]$. A full reconstruction of the original data therefore requires this scale factor to be known, in addition to the encoded form of the trace. However, we cannot simply set $f_0 = -1, f_1 = 1$. We must recognize that recorded seismic data contains a significant random noise component, which may increase or decrease the signal at any particular point in time. It is therefore possible that the 'true' seismogram—which we are seeking to represent in our reduced-dimension space—may have a greater maximum amplitude than that recorded. We, therefore, choose f_0 and f_1 to span a moderately larger range than the data; typically, we use $f_0 = -1.1, f_1 = 1.1$.

As things stand, the training algorithms set out in the previous section assign equal significance to each datum, and attempt to fit all equally well. We note that the average power in a signal $s(t)$ is defined

$$\bar{P} = \frac{1}{T}\int_0^T |s(t)|^2 \, \mathrm{d}t\,, \tag{31}$$

or, when this is digitized into N samples $\{s_1, s_2, \ldots, s_N\}$,

$$\bar{P} = \frac{1}{N-1}\sum_{i=1}^{N} s_i^2\,. \tag{32}$$

A comparison of eqs (17) and (32) makes it clear that the training algorithms seek to minimize the power of any parts of the training data that may be lost in the encoding–decoding process. Because our assertion is that this unrepresented portion of the data corresponds largely to the noise component, we should ensure that the noise power is broadly equivalent between traces. This is somewhat

difficult to quantify, but it is clear that the scaling of individual waveforms will distort matters. By giving all traces the same maximum amplitude, we effectively increase the significance of the noise in those traces that originally had a lower amplitude, and the training algorithm will attempt to learn this at the expense of 'useful' signal in the higher amplitude traces. To counteract this, we introduce a weight into the various learning rules. If we attach a weight ϕ_j to the jth trace, the error function defined in eq. (17) becomes

$$E = \frac{1}{2} \sum_{i,j} \phi_j \left(x_{ij}^{(L)} - x_{ij}^{(0)} \right)^2 \tag{33}$$

and the network parameter update rules (eqs 21–23) become, straightforwardly

$$b_i^{(n)} \rightarrow b_i^{(n)} - \frac{\eta}{Q} \sum_j \phi_j \Delta_{ij}^{(n)} u_{ij}^{(n)} a_i^{(n)} , \tag{34}$$

$$W_{ij}^{(n)} \rightarrow W_{ij}^{(n)} - \frac{\eta}{Q} \sum_k \phi_k \Delta_{ik}^{(n)} a_i^{(n)} u_{ik}^{(n)} x_{jk}^{(n-1)} , \tag{35}$$

$$a_i^{(n)} \rightarrow a_i^{(n)} - \frac{\eta}{Q} \sum_j \phi_j \Delta_{ij}^{(n)} u_{ij}^{(n)} \left(b_i^{(n)} + \sum_k W_{ik}^{(n)} x_{kj}^{(n-1)} \right) . \tag{36}$$

For the CRBM learning rules (eqs 27–29), we simply make use of a weighted mean, so that

$$\langle \chi \rangle = \frac{\sum_i \phi_i \chi_i}{\sum_i \phi_i} . \tag{37}$$

The most obvious choice of weighting is one derived from the original trace amplitudes, although this may depend on the details of the data set, and on the intended applications of the trained autoencoder.

3.3.2 Adaptive learning rates

All learning rules presented in the previous section (eqs 21–23; 27–29; 34–36) feature a learning rate parameter, η. In general, there is no requirement for η to take the same numerical value in all cases; it may prove desirable to encourage the system to adjust one class of parameters more rapidly than another. The purpose of η is to control the size of the step the algorithm makes at each iteration. If too small a step is taken, progress towards convergence may be very slow; too large a step may prevent the system exhibiting convergent behaviour at all. We therefore wish to find an optimum learning rate, that balances these two considerations.

This is particularly relevant for the autoencoder training stage, where the learning rules are derived from the local gradient of the error function (eq. 17). Because we use only first-order derivatives, we are essentially making a linear approximation to the error function, and η must therefore be small enough that this is valid. The controlling factor is thus the local curvature of the error function, which may vary as training progresses. This can cause problems: a network that has been learning successfully for several hundred iterations may encounter a region of greater local curvature, leading to rapid divergence. One solution, of course, is to always use an extremely low value of η; however, the increase in training time makes this impractical. Our approach is unsophisticated, but straightforward. We make the transformation

$$\eta \rightarrow \eta_0 + \frac{i}{I} (\eta - \eta_0) , \tag{38}$$

where η_0 represents the minimum learning rate to be used, and η now specifies the *maximum* learning rate. Initially, i is zero, and is incremented by one on each iteration that leads to a reduction in the error as defined by eq. (17), up to a maximum value of I. If an iteration increases the error according to that measure, we halve i. We find this is usually sufficient to allow the training algorithm to stabilize; however, we also monitor the frequency with which such action is necessary, in order to identify cases where η is set too high. With appropriate choices for the various parameters, this approach allows us to achieve a reasonable balance between the competing requirements of speed and convergence.

3.3.3 Initialization and typical settings

Before training commences, all network or CRBM parameters must be randomized, to allow individual neurons to focus on different aspects of the data set. Typically, we choose

$$w_{ij}^{\mathrm{v,h}} = \mathcal{G}(0, 0.01) ; \qquad b_i^{\mathrm{v,h}} = \mathcal{G}(0, 0.01) ; \qquad a_i^{\mathrm{v,h}} = 1 , \tag{39}$$

where, again, $\mathcal{G}(\mu, \sigma)$ represents a random sample drawn from a Gaussian distribution. There is no great significance to the values chosen. However, because the weights are used to generate the argument of an exponential function, overflow errors may be encountered if the typical magnitude of these are too large. The standard deviation of the distribution used in initialization must be chosen with this in mind.

As a guideline, we use learning rates in the range $0.1 \lesssim \eta \lesssim 0.3$, and we typically use $\eta_0 = 0.01$. Similarly, we find that $I \approx 100$ in eq. (38) leads to satisfactory behaviour. However, as we have already suggested, appropriate values for the learning rate depend on the detail of the data sets in use; as a result, some trial and error may be required.

4 DEMONSTRATION

To demonstrate the power of autoencoder networks in conjunction with seismic data, we begin with a quite general example, using long-period surface wave data. From an initial data set containing seismograms recorded by the IRIS/IDA global networks for all events with magnitude $M_{\mathrm{W}} \geq 6.5$ in 2000, we extract 6000 traces at random, with the restriction that the source-receiver epicentral distance should lie in the range $40^\circ \leq \theta_E \leq 140^\circ$. These are filtered to enhance the surface waves (cosine bandpass filter, corner frequencies 1.0 mHz, 2.0 mHz, 6.7 mHz, 7.4 mHz), and inspected visually to discard poor quality traces. The result is around 1700 waveforms judged to be of high quality; from these we form a 1000-element 'training set', and a 500-element 'monitoring set'. Some brief statistics concerning the composition of these can be seen in Fig. 6. Caveats concerning the creation of a 'visually clean' data set were discussed in Valentine & Woodhouse (2010), and they apply equally here: principally, that not all classes of 'error' in seismic data may be detected by a simple visual inspection, and that any such classification is inherently a subjective process.

The highest frequency component retained in the data after filtering has period $T = 135$ s. Following Nyquist (1928), a full reproduction of this signal requires it to be sampled at least once every 67.5 s. Given an original sampling rate of 1 Hz and taking into account the efficiency of power-of-two downsampling strategies (e.g. Cooley & Tukey 1965), we convert our data set to a sampling frequency of 62.5 mHz—one sample every 16 s. Each trace is truncated after 512 data points—equivalent to just over 2 hr of seismic data per record, beginning at the event time. In consequence, our data set is sampled

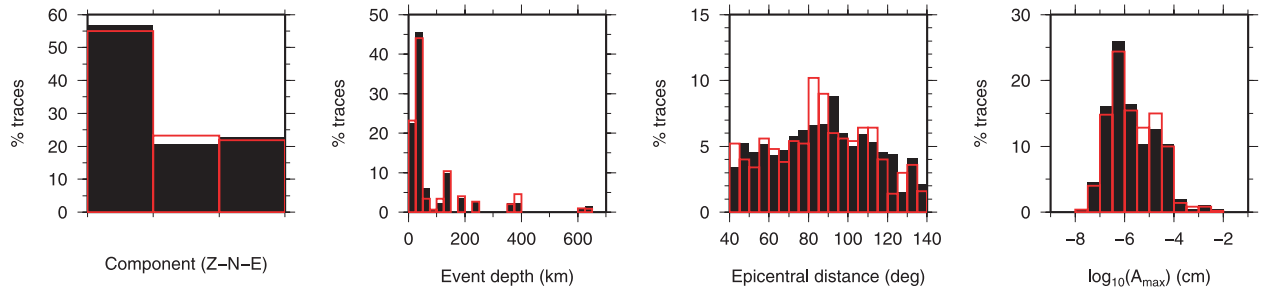

Figure 6. Composition of data sets. Histograms showing make-up of training (solid black bars) and monitoring (red outline) data sets, by orientation, event depth, epicentral distance and maximum trace amplitude. Visually good-quality data sets were derived from random samples of seismograms from all events of $M_W > 6.5$ in 2000. Typically, we find horizontal-component traces to have a lower signal-to-noise ratio than vertical components, explaining the apparent over-representation of the latter in the data sets. Note that all traces are normalized to have unit maximum amplitude before network training.

roughly four times more densely than necessary on purely spectral grounds. All traces are scaled by their maximum amplitude, so that the data lies in the range $[-1, 1]$.

We use this data set to create a 512-256-128-64-32-64-128-256-512 autoencoder. This represents a compression ratio of 16× compared to the original data, 4× better than expected based on Nyquist's theorem. Our decision that the number of nodes in adjacent layers should be related by a factor of two is based on experience; a broad justification for this can be made on the basis that the logistic function behaves as a (fuzzy) binary switch. For this example, we initialize all weights as set out in eq. (39), and have a CRBM learning rate $\eta^{(C)} = 0.3$; the maximum autoencoder learning rate is $\eta_0^{(A)} = 0.1$. For the time being, we choose not to introduce any noise during training of either CRBM or autoencoder.

We begin by generating four CRBMs, and training each of these for 500 iterations; these are used to assemble an autoencoder and a further 2500 iterations of training are conducted: at this point, the reconstruction error for the monitoring data set has ceased to change significantly. Once training is complete, we use the network to encode the 500-element monitoring data set; some examples of waveforms and their corresponding encodings can be seen in Fig. 7. These may then be decoded, and compared with the original traces to assess the success of the autoencoder representation. To quantify the error, we make use of the same error function used during network training, eq. (5), computed on a per-trace basis.

Histograms showing the distribution of errors across the training and monitoring data sets may be seen in Fig. 8, and some examples of input and recovered waveforms—best, worst and intermediate—can be found in Fig. 9. Overall, results are promising: the majority of waveforms appear to pass through the encoder–decoder process relatively unchanged, and errors are the result of small alterations throughout the trace, rather than large, concentrated anomalies. A relatively small number of traces are found to have larger error values, and it appears that these typically arise from the loss of some particular wave packet in the original data. Notably, the network demonstrated here appears not to be recovering large-amplitude signals at the very beginning of the waveform: this arises because such signals occur rarely in our training data set. This highlights the potential for the the autoencoder to be used as a 'novelty detector'—features that are not characteristic of the training data set will tend to be poorly recovered after encoding. Depending on circumstance, a seismologist might wish to use this information to discard anomalous data, or target 'interesting' signals for further investigation. We provide a more concrete example of this at the end of the current paper.

Fig. 10 shows histograms of the elements of $\mathbf{W}^{(n)}$, $\mathbf{b}^{(n)}$ and $\mathbf{a}^{(n)}$ from eq. (14), on a layer-by-layer basis, for our trained autoencoder. Overall, we find that the weights and biases are distributed around zero, as we expect. It is interesting to note that layer 5, the first 'decoder' layer, has both a larger range of values of weights than most other layers, and a sensitivity that is almost double than that found in the 'encoder' layers. The effect of this is to make it more likely that the outputs from this layer will be close to saturated.

4.1 The encoding domain

As can be inferred from Section 3, the mapping between encoding and waveform is non-linear, and developing any intuition for its behaviour is not straightforward. To provide some appreciation for the behaviour of the network, Fig. 11 shows the waveforms generated from the unit encodings $\mathbf{x}^{(L/2)} = (1, 0, 0, \ldots)$, $\mathbf{x}^{(L/2)} = (0, 1, 0, \ldots)$, etc. We observe that the resulting traces could themselves be plausibly classified as seismograms; each element of the encoding clearly contains information from throughout the seismogram, and there is no temporal localization. The aforementioned inability to represent signals at the very start of a trace may also be clearly seen.

In Fig. 12, we show the 32 × 32 inner product matrix

$$M_{ij} = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{\mathbf{x}_i \cdot \mathbf{x}_i}, \tag{40}$$

computed using the waveforms from Fig. 11. We observe that these 32 waveforms are relatively close to forming an orthogonal set. By construction, the unit encodings are orthogonal, and these relationships are approximately preserved during the decoding process. This is indicative of a 'well behaved' transformation, with the departure from true orthogonality indicative of the degree of compression within the system.

4.2 Stability

Under these circumstances, we should consider two (related) questions. First, how stable is the encoder–decoder pair with respect to small perturbations to the inputs? A minor noise level variation in an input seismogram should not result in its appearance changing markedly after passing through the network. Following on from this: if two seismograms are 'close' in the time domain, are they similarly 'close' in the encoding domain? This question is significant, because one potential application of the autoencoder method lies in the reduction of the dimension of the data space for non-linear tomographic inversions (e.g. Snieder 1998; Meier *et al.* 2007). Typically, such methods are extremely computationally intensive, and any possibility of reducing data set volume may be beneficial.

The first may be approached by considering the derivatives of output layer with respect to input layer. Following eq. (A11), we can

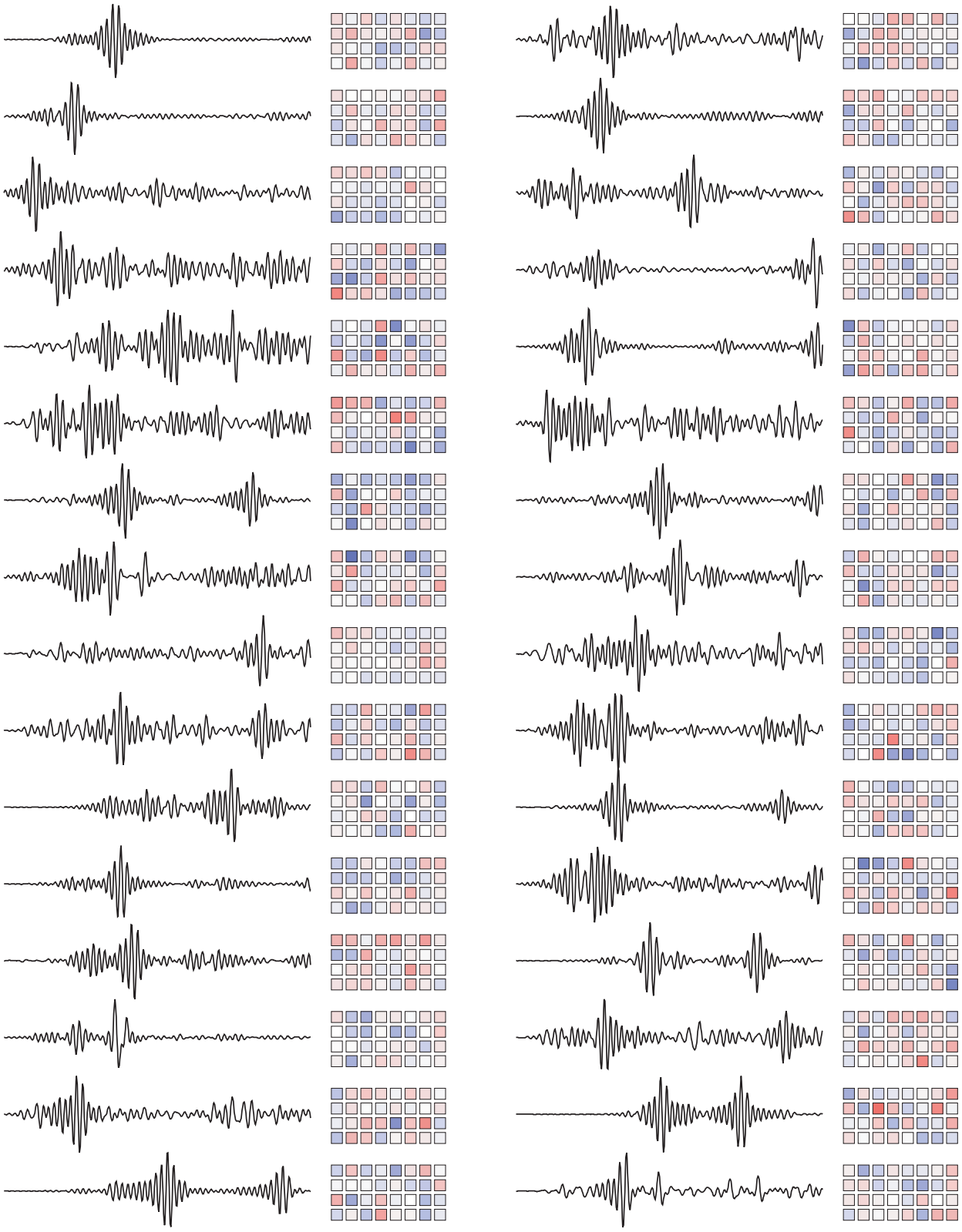

Figure 7. Examples of encodings. Waveforms from the monitoring data set and their encoding using a trained autoencoder. As there is no inherent temporal relationship between the elements of each encoding, we represent them in grid form; red colours represent positive numbers, and blues are negative. The colour scale is equivalent to that shown in Fig. 12.

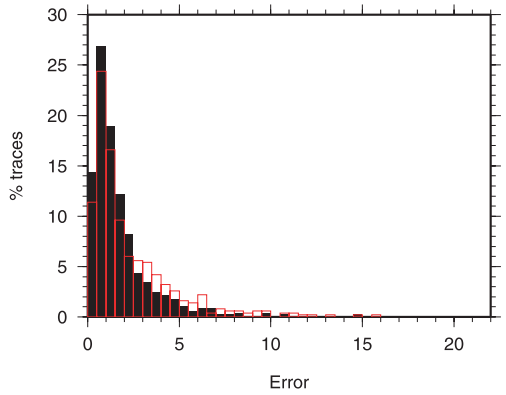

Figure 8. Errors in recovery of original traces from encoding. Histogram showing trace-by-trace error computed according to eq. (5) for training (solid black bars) and monitoring (red outline) data sets. Some examples from the monitoring data set can be found in Fig. 9, to allow error levels to be related to waveform discrepancies.

define the matrix $\mathbf{D}^{(n)}$ such that

$$D_{ij}^{(n)} = \frac{\partial x_i^{(n)}}{\partial x_j^{(n-1)}}\,, \tag{41}$$

$$\mathbf{D}^{(n)} = \mathbf{a}^{(n)} * \mathbf{u}^{(n)} * \mathbf{W}^{(n)}\,. \tag{42}$$

From the chain rule, we then see that if we choose to define $\mathfrak{D}$ to represent the derivative of outputs with respect to inputs,

$$\mathfrak{D}_{ij} = \frac{\partial x_i^{(L)}}{\partial x_j^{(0)}}\,, \tag{43}$$

$$\mathfrak{D} = \mathbf{D}^{(L)}\mathbf{D}^{(L-1)}\cdots\mathbf{D}^{(1)}\,. \tag{44}$$

Thus, $\mathfrak{D}$ can be computed straightforwardly for any given input vector $\mathbf{x}^{(0)}$ (as $\mathbf{u}$ depends on $\mathbf{x}$). Ideally, of course, we wish the result to be close to an identity matrix.

Three examples of $\mathfrak{D}$ are shown in Fig. 13. In all cases, we see that sensitivity is concentrated close to the diagonal: that is, an infinitesimal change in some point of the input seismogram leads to infinitesimal changes in some small region close to the corresponding point in the reconstructed seismogram. Some 'blurring' is perhaps to be expected given the compression in the system; furthermore, the relationship between adjacent points is an important feature of seismic data, and it is unsurprising that the network displays evidence of such connections.

To answer the second question, we must begin by defining a measure of 'closeness' of two vectors; we continue to use that used during network training, eq. (5). We generate waveforms close to members of the monitoring data set by the addition of random noise, and encode these using the trained autoencoder. We can then compare the error between original waveforms to that between the encodings; some results are shown in Fig. 14. Again, the waveform difference associated with a given error can be understood, qualitatively, from Fig. 9. There is a clear relationship: the distance between two encodings varies linearly with the distance between the original waveforms, so that any change that acts to bring two encodings closer together also brings the corresponding waveforms into agreement. This is an important and promising result for tomographic applications.

4.3 Variations

The preceding sections have focussed on a particular autoencoder, and certain choices of the various parameters that control network behaviour. Indeed, it is worth noting that because the network initialization is random, the course taken during training will vary between networks trained on the same data set with the same choices. The end results may not be numerically identical, and we would not expect two distinct networks to yield the same encoding for a given waveform. Nevertheless, the general properties and characteristics set out in the previous section are robust, and routinely emerge during network training. We emphasize that although training has a stochastic component, the operation of the trained encoder–decoder system is deterministic.

4.3.1 Learning rates and noise

In Fig. 15, we show the iteration-by-iteration evolution of the error function (eq. 17) during autoencoder training for a number of different choices of learning rate and noise level in both CRBM and autoencoder training phases. In the main, the behaviour is intuitive, although it is important to distinguish between the effects on training and monitoring data sets. Parts (a) and (b) show the effect of altering the CRBM and autoencoder learning rates; as expected, a greater learning rate leads to a more rapid reduction in training data set error. However, these also illustrate the phenomenon of 'overfitting'—at some point during training, the network starts to learn the intricacies of the training data set, at the expense of its ability to represent unseen examples. To prevent this, we can inject noise into the data set during training.

Parts (c) and (d) show the effects, at the autoencoder stage, of having introduced noise during CRBM training. Clearly, the CRBM noise is beneficial to the autoencoder's ability to learn the detail of the training set, presumably by promoting a more complete exploration of the parameter space, but seems not to impact significantly on the ability to represent the monitoring data set. In the light of what we shall see in (e), (d) repeats (c) with a lower learning rate but more training iterations; however, the consequences are limited.

Parts (e) and (f) show the effects of introducing noise during autoencoder training. The training set errors are increased, because the noise now contributes; however, in (e) we see that the training process has become unstable. This can be fixed, as in (f), by reducing the overall learning rate; this limits the influence stochastic effects can have on the network. The principal benefit of noise during autoencoder training is as a form of regularization (see, e.g. Bishop 1995): it acts to desensitize the network to small-scale features within the training set, and prevents these being 'learned' at the expense of an ability to represent more general data sets.

Perhaps the general conclusion to draw from Fig. 15 is that autoencoder training is relatively stable for sensible choices of the various parameters controlling the algorithm. The introduction of some noise, at least during autoencoder training, appears beneficial, although this may require lower learning rates than would otherwise be possible. We note that the manner in which noise is introduced differs between CRBM and autoencoder training, so that a direct comparison of numerical values of σ in each case is not useful. Indeed, appropriate noise levels may well vary between data sets and, perhaps, intended uses for the encodings. As a result, an analysis of appropriate parameters should be undertaken when setting up an autoencoder problem.

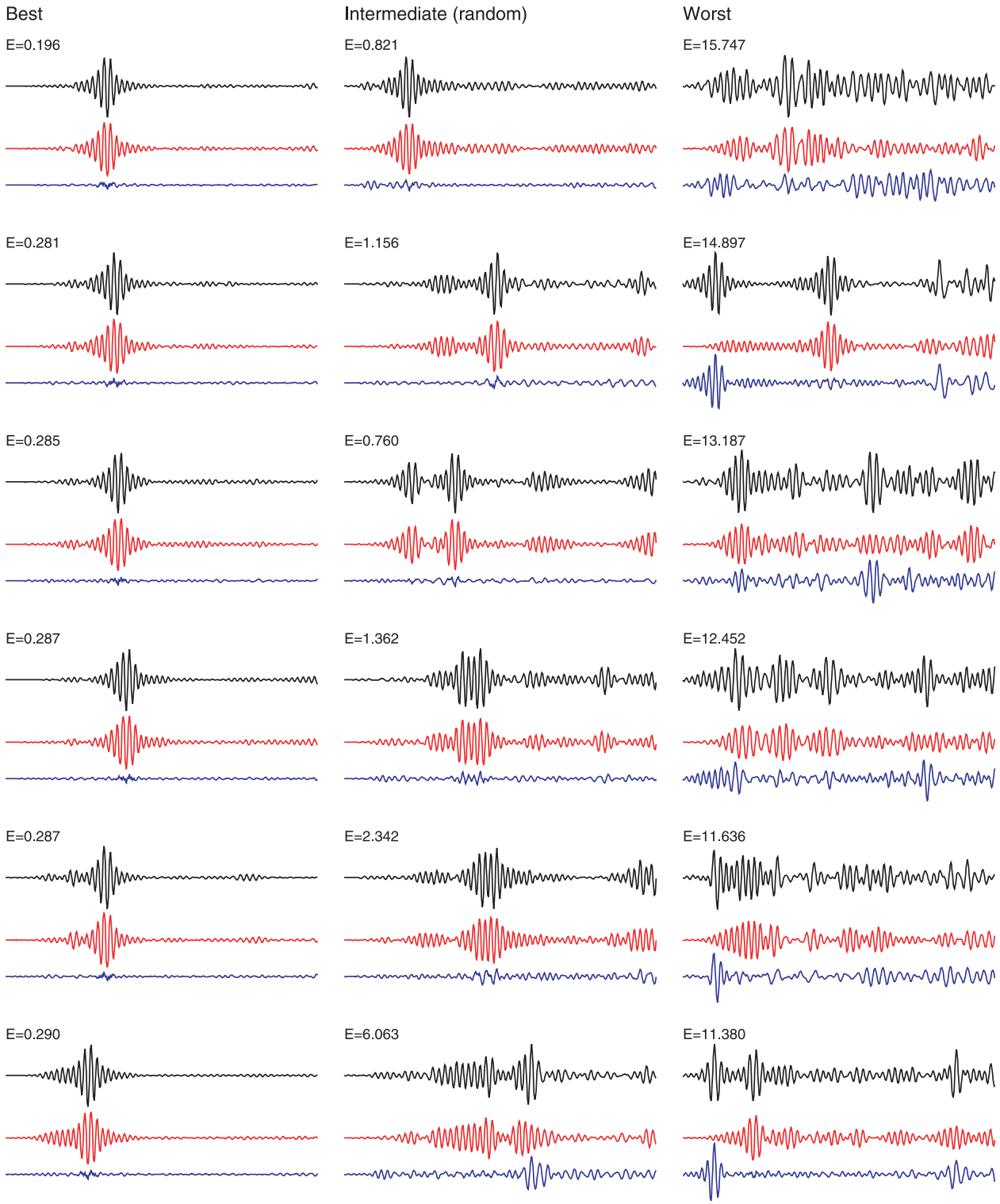

Figure 9. Demonstration: original and decoded traces. 512-point seismograms (black) from the 'monitoring' data set encoded to 32-element representations, and then recovered (red). Blue lines show difference between input and recovered traces; all are plotted using the same vertical and horizontal scales. The recovery error is quantified according to eq. (5); the best (left column) and worst (right column) examples are shown, along with a random selection from the remainder of the data set.

4.3.2 Compression ratios

In the examples presented here, we have used a compression ratio of 16x (512 → 32) against the raw data, around 4x better than achievable by sampling the traces at their Nyquist frequency. In principle, it would be instructive to explore the extent to which seismic data can be compressed, and how recovery error trades off against compression ratio. However, to do so in a robust manner is complex, because any change in the number of encoding-layer

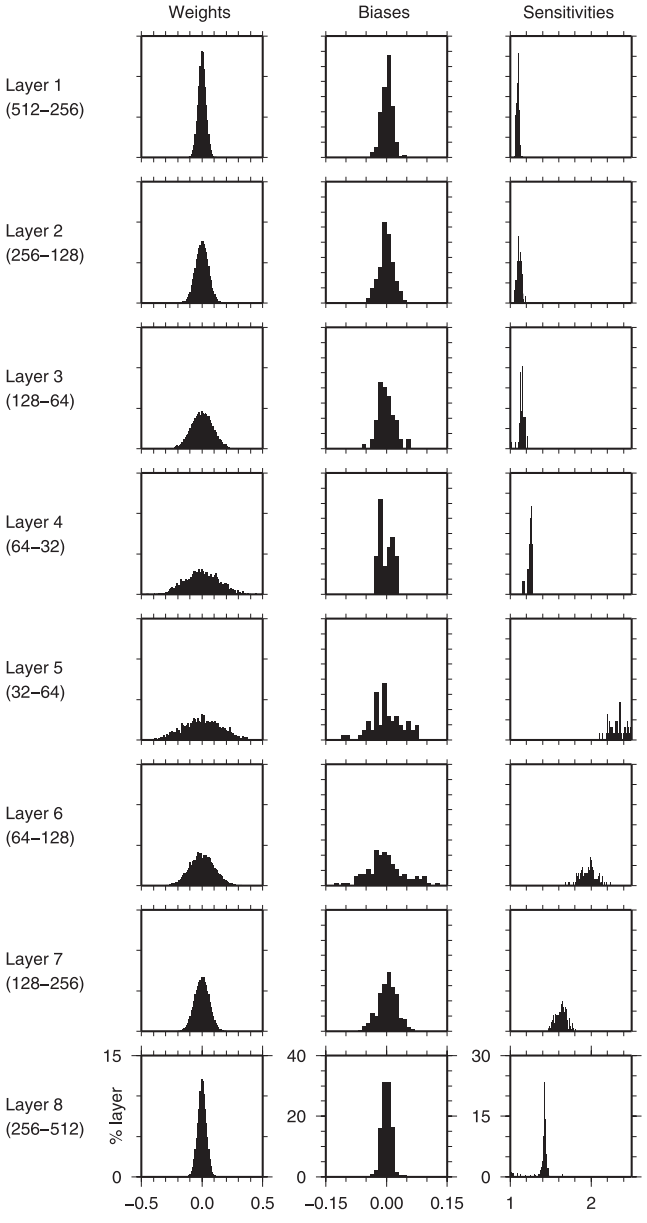

Figure 10. Weights, biases and sensitivities of trained autoencoder. Histograms showing distribution of values of elements of $\mathbf{W}^{(n)}$, $\mathbf{b}^{(n)}$ and $\mathbf{a}^{(n)}$ (as defined in eq. 14) for a trained autoencoder. All histograms in a given column share the same horizontal and vertical scales. Note that the number of elements varies between layers.

nodes entails significant changes to the overall network architecture, raising questions about the comparability of results.

Halving or doubling the compression ratio can, in principle, be achieved by altering the number of layers in the network. However, this may affect the complexity of the mapping representable by the network (*in extremis*, consider reducing the network to contain only one layer in encoder and decoder), so that any observed effects are not attributable only to changes in compression. Similarly, changes effected by simply adding or removing nodes within the existing layers alter the 'factor of two' relationship, which we have suggested may be appropriate in conjunction with logistic neurons. Such effects must be considered as part of any study of seismogram compressibility, and are beyond the scope of the current work. Nevertheless, we suggest that such a study may well be worthwhile, because it could throw much light on the information content of seismic data, with implications for studies of the seismic source, and of earth structure.

4.3.3 Network architecture

We have, of course, only made use of a single network architecture: the structure set out in Section 3 can be altered in many different ways. In particular, neurons with different properties may be used, and there is no *a priori* requirement for all neurons in the network to be of the same type. Equally, there are many alternative choices of error function (eq. 17), which would alter the training behaviour of the network. We have not explored these possibilities, and do not claim that the formulation set out here is the only—or best—means of implementing autoencoders for seismic data.

4.3.4 Data

Similarly, we have made use of only a single-data type for our examples. However, there is no reason why the method should not be applicable to any seismic data set—although, as alluded to earlier, it is likely that the compressibility of data sets will differ, according to the complexity and range of signals to be be represented. It is also likely that different classes of data may require different choices of parameters: the learning of more complex data sets might, for example, be aided by a lower learning rate. We reiterate our previous comment, that such effects should be explored on a per-case basis.

5 DISCUSSION AND APPLICATIONS

The autoencoder method set out and demonstrated here appears to provide a powerful technique for discovering lower-dimensional structure within complex waveform data sets. The lower-dimensional representations appear to behave 'well', as discussed in the previous section: the evidence presented here seems to suggest that analysis typically conducted in the time domain may be equally possible in the encoding domain. Of course, the benefits and drawbacks of such a transformation must be assessed in the context of each particular case.

The behaviour of an autoencoder network is strongly dependent on the training data set used, because this implicitly defines $\mathbb{S}^N$, and an appropriate choice for this may be instrumental in the success or failure of any given situation. It is important to bear in mind that the network can only properly be used in conjunction with waveforms that meet whatever selection criteria—both explicit, and implicit—were used for the training set. By way of illustration: we might expect to be able to represent seismograms from some local seismic array tolerably using a network trained using data from around the globe. However, we would not expect to be able to represent global data sets using a network trained using data from a particular array. Conversely, the more we can restrict the training set, the better results are likely to be: for a given compression ratio, seismograms from the array are likely to be better represented by a network trained only for that array than by a similar network trained globally.

As stated in the introduction, we do not foresee the autoencoder finding common use for the compression of seismic data for storage or transmission. As we have seen, recovery is not perfect—although we suggest that where seismograms contain moderate amounts of noise, the recovered trace might be as plausible a representation of the true ground motion as the original—and modern advances in digital storage technology make the need for compression less

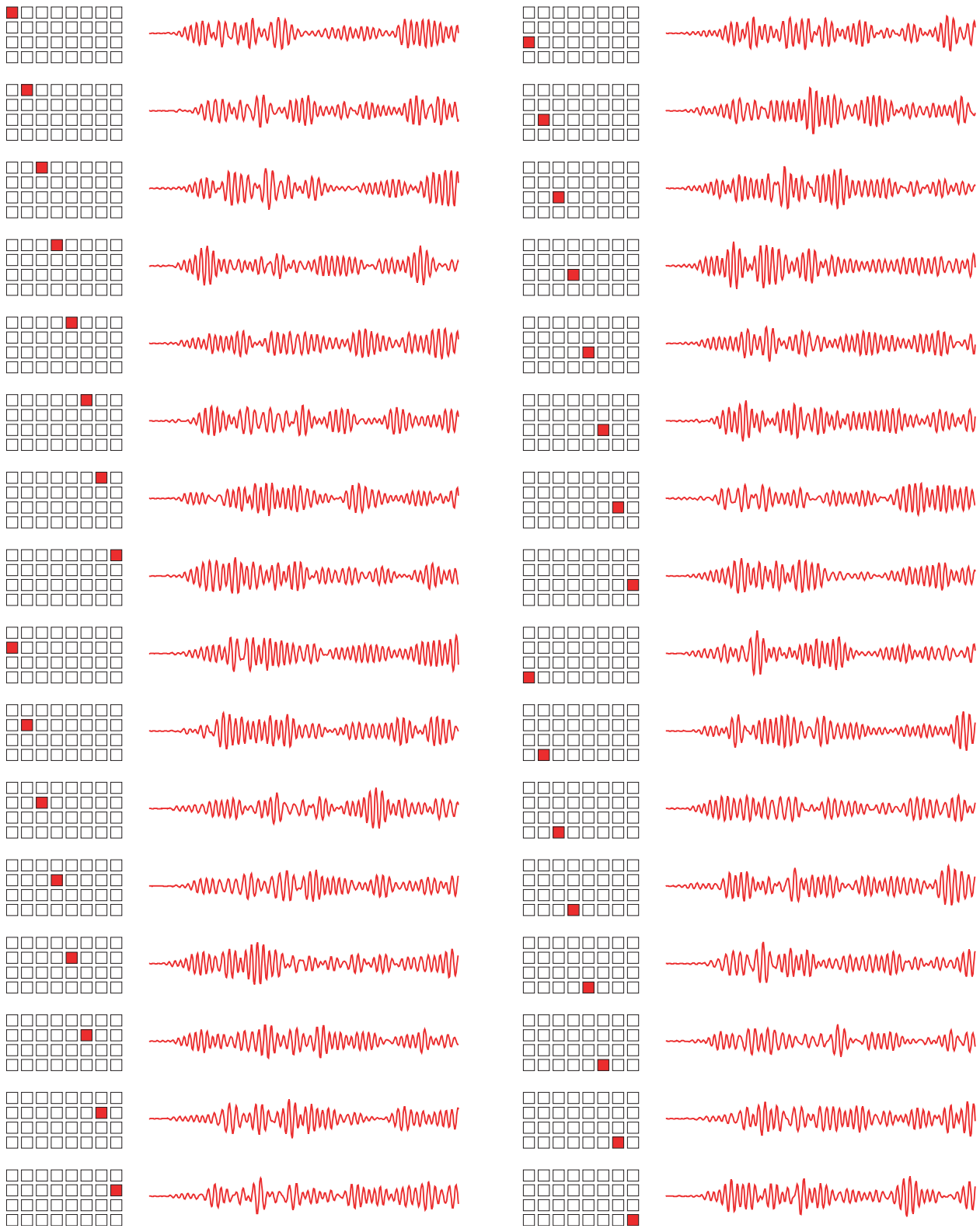

Figure 11. Waveforms corresponding to unit encodings. 32 waveforms obtained by decoding the unit vectors $\mathbf{x}^{(L/2)} = (1, 0, \ldots, 0)$, $\mathbf{x}^{(L/2)} = (0, 1, \ldots, 0)$, etc. We observe that each element of the encoding affects the entire length of seismogram.

pressing. Instead, we foresee three broad categories of application for the autoencoder method: as a technique for analysing waveform data, and the information contained therein; as a means of reducing data set dimension, so that encodings are used as a direct proxy for the waveforms themselves and as a practical tool for the management of extremely large data sets. Of course, the boundaries between these classes are not always clear-cut, and some applications may not fit neatly into one particular category.

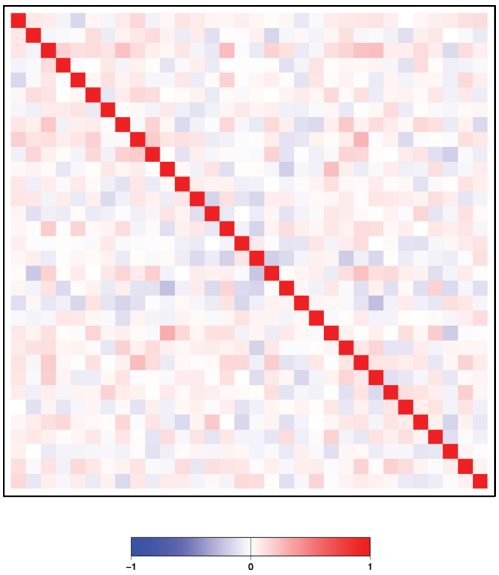

Figure 12. Orthogonal encodings produce (almost) orthogonal decodings. Inner product matrix **M** (as in eq. 40) formed from the 32 waveforms shown in Fig. 11. Clearly, these are close to forming an orthogonal set.

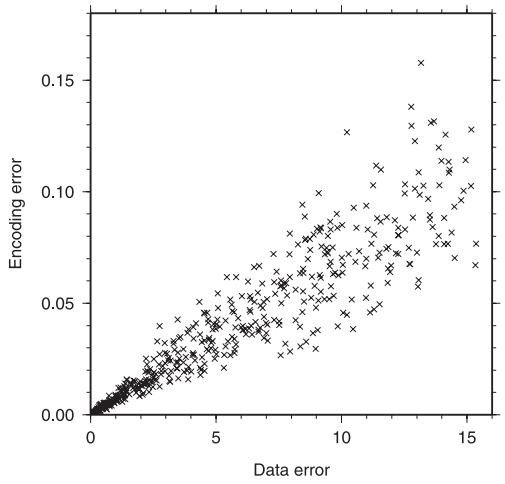

Figure 14. Waveforms that are 'close' have encodings that are 'close'. Plot of error between two waveforms against error between the corresponding encodings, defined as in eq. (5). For the interpretation of a given error in terms of waveform discrepancies, see Fig. 9.

5.1 Analysis of waveforms

Clearly, the autoencoder assimilates information about the features present within individual waveforms, and their relationships with one another. Further, the relative compressibilities of different classes of data is related to the relative information content—in a mathematical sense—of those classes. As a result, it should be possible to use autoencoders to develop a greater understanding of how the various parts of a waveform are linked, with attendant benefits for studies that make use of full-waveform data: we may gain a greater understanding of how to enhance sensitivity to particular aspect of earth structure or source mechanism.

In a similar vein, we should consider the significance of the 'basis' waveforms discovered by the network, such as those shown in Fig. 11. If these do contain some underlying information about the overall structure of the waveforms in our data set, it may be possible to use these in conjunction with classical methods of filter design to improve tomographic resolution. Clearly, such possibilities have not been proven on the basis of the current work, but merit further exploration.

5.2 Operations on encoded waveforms

In general, seismic data is used to draw inferences—about sources, earth structure or both. Given that the encodings themselves are sufficient to allow the reconstruction of the original waveforms, it is, in principle, possible to analyse such problems in the encoding domain. This may be desireable if it either enhances sensitivity, or reduces computation time. The former may arise because some system decouples in the encoding domain; the latter will typically simply exploit the reduction in the number of parameters to be handled.

As mentioned above, we envisage this being particularly significant for non-linear tomographic studies using neural networks. Such approaches are currently in their infancy, but would typically involve training a neural network to associate earth model parameters with synthetic waveforms computed in that model. In principle,

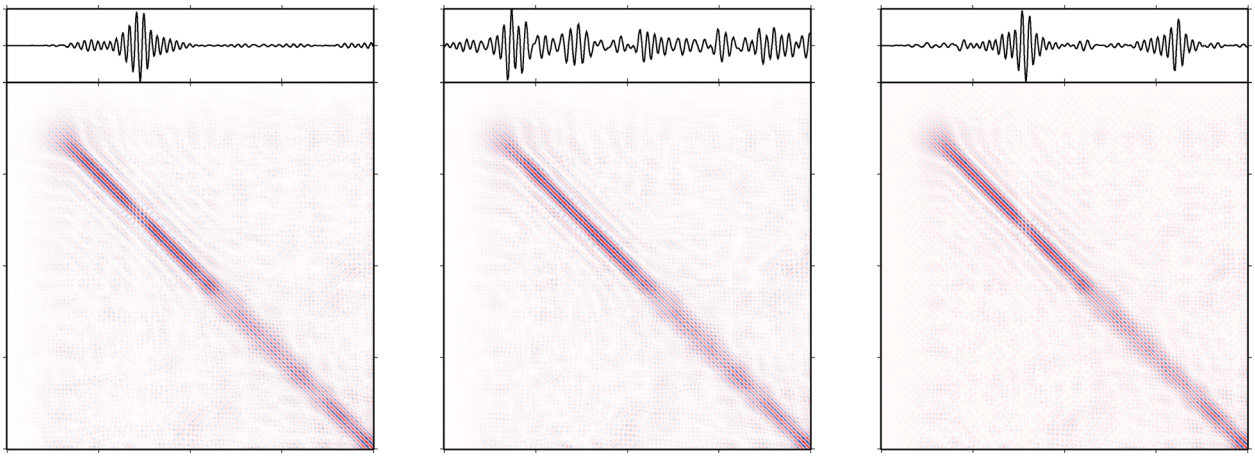

Figure 13. Derivatives of network outputs with respect to network inputs. Plots of the matrix $\mathfrak{D}$ (eq. 43) corresponding to three different seismograms (as shown). In all cases, off-diagonal elements of $\mathfrak{D}$ are close to zero, so that there is a strong localization of sensitivity within the network—a small change to the input seismogram at time t may be reflected in the output seismogram in some region close to time t, but has negligible impact far from this. Matrices are normalized by their maximum element.

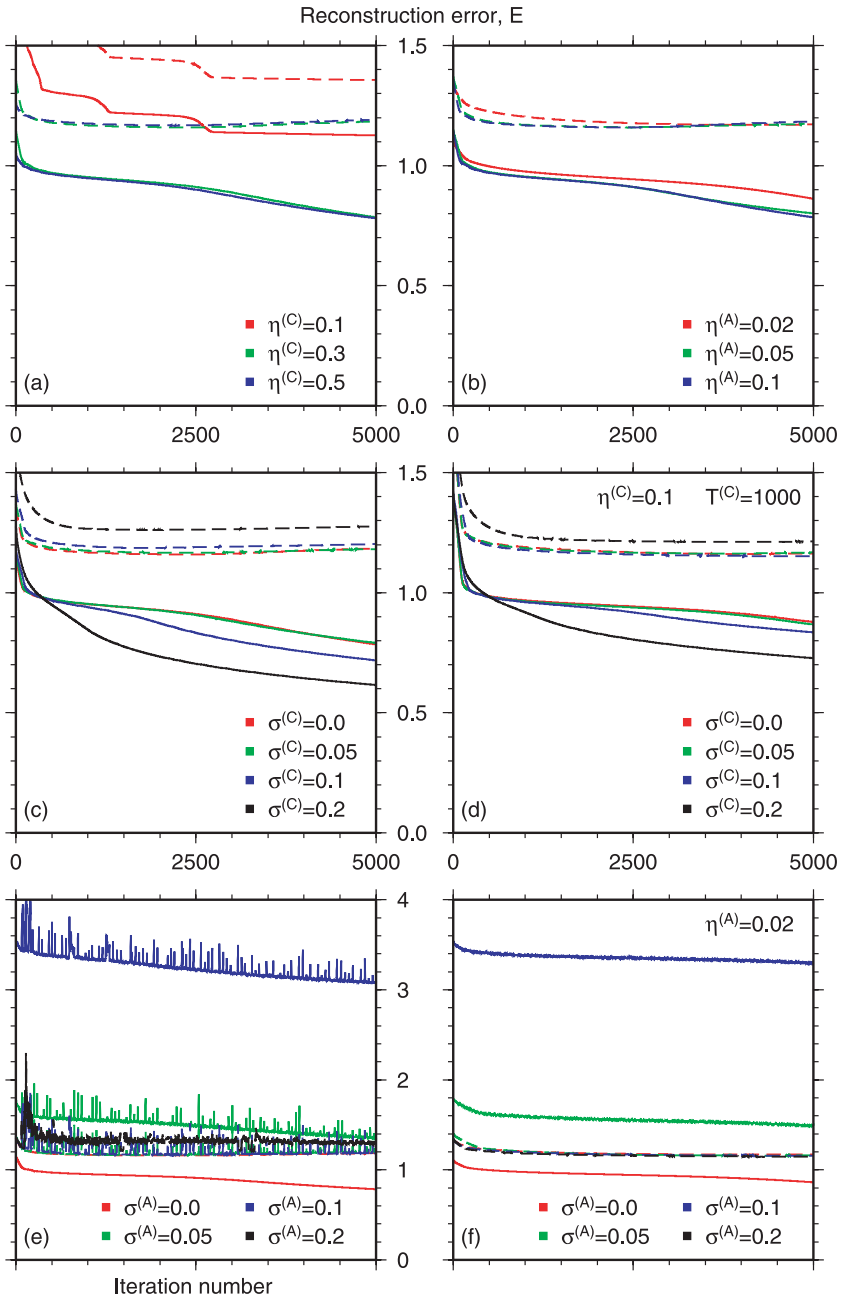

Figure 15. Effect of learning rate and noise on training. Iteration-by-iteration change in overall error (as in eq. 17) during 5000 iterations of autoencoder training, for training data set (solid lines) and monitoring data set (dashed). (a) and (b): Increasing learning rate causes error to decrease more rapidly; (c) and (d): Noise in CRBM training improves representation of training set, but not generalization; (e) and (f): Noise in autoencoder training prevents overfitting of training set, but introduces instabilities. For more complete description, see main text. Where not otherwise specified, parameters were based on those in the 'demonstration' (Section 4): learning rates $\eta^{(C)} = 0.3$ and $\eta_0^{(A)} = 0.1$; noise levels $\sigma^{(C)} = 0.0$ and $\sigma^{(A)} = 0.0$ (i.e. no noise), number of training iterations $T^{(C)} = 500$ and $T^{(A)} = 5000$.

the parameters of the real Earth can then be inferred from recorded data. However, as will be appreciated from the networks discussed in this paper, the number of weights required to specify such a network will grow rapidly with the number of waveform parameters to be represented. Thus, even a moderate reduction in this can have important consequences for the tractabilty of such inversions.

It may also be possible to use encodings to detect particular features, or search for given occurrences in waveform catalogues. In principle, such features might have a better-defined signature in the encoding domain than in the time domain; once this is identified, it is straightforward to construct a classifier that determines whether individual seismograms are likely to contain the feature. Such applications are likely to make use of a modified error function during network training, to promote and enhance the desired effects.

5.3 Automated handling of large data sets

One of the key challenges in modern seismic studies is the sheer quantity of data that must be handled. Typically, raw data sets contain

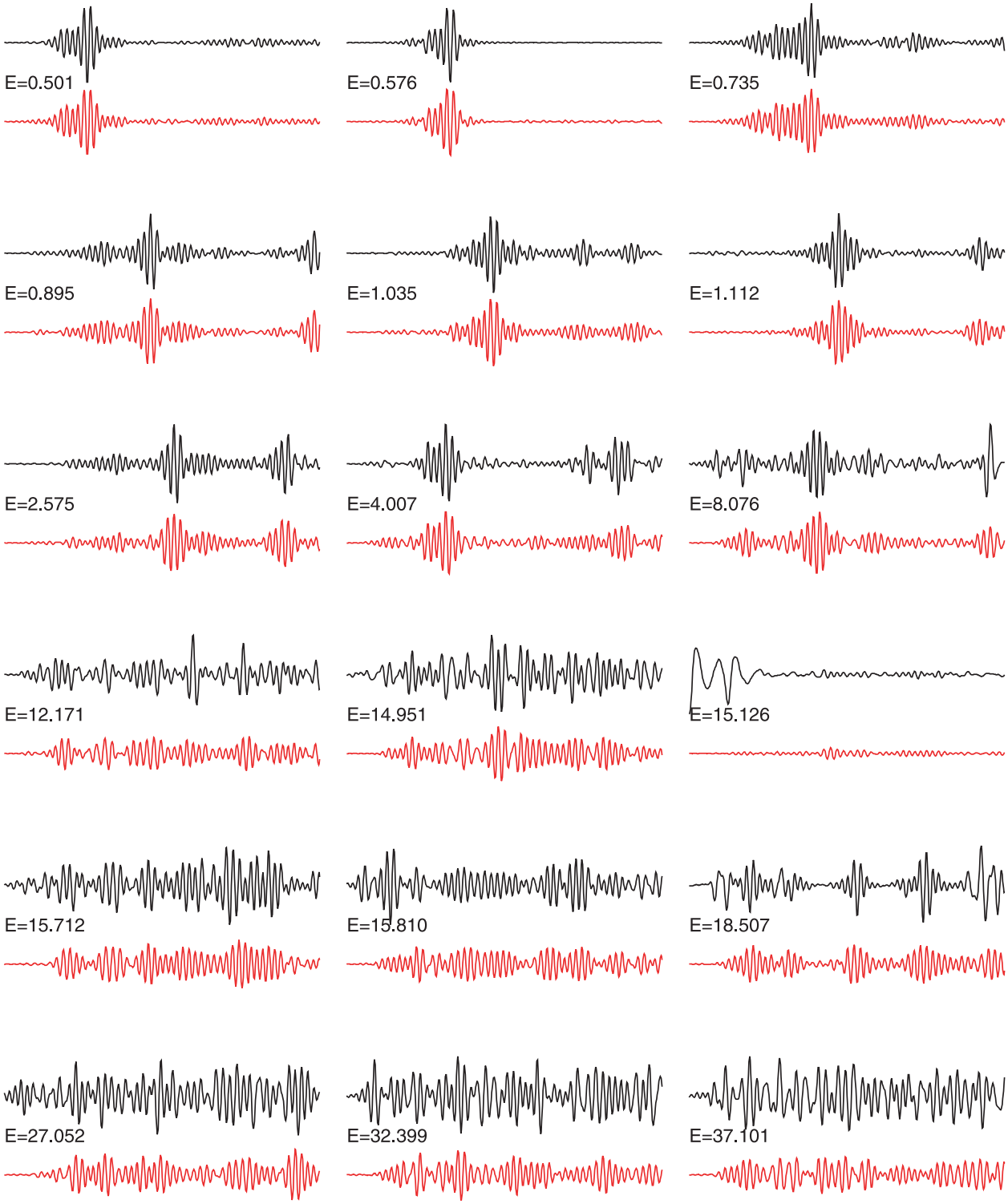

Figure 16. Reconstruction error as a measure of waveform quality. An autoencoder trained on a data set of hand-picked, high-quality surface wave seismograms is used to encode and reconstruct similar seismograms in a previously unseen data set. The reconstruction error, computed according to eq. (5), provides a measure of how well each trace conforms to the characteristics of the training set. We present a random sample of waveforms (black) and their reconstructions (red), sorted according to reconstruction error. There is a clear correlation between visual quality, and the ability of the network to recover the original trace. Poor quality traces can therefore be removed from the data set by discarding those over some reconstruction threshold—see Fig. 17.

a wide variety of flaws—noise, recording glitches, instrument errors and so forth. Some of these may be detected and eliminated by rule-based methods; however, in many cases, there is little alternative to a visual inspection of all waveforms. Such an approach is extremely time-consuming, and limits the size of data set that can feasibly be used.

In recent years, some attempt has been made to construct systems that may introduce a degree of automation to this process (e.g.

Maggi *et al.* 2009; Valentine & Woodhouse 2010). Autoencoders provide a new approach to such problems, through the 'test for membership of $\mathbb{S}^N$' set out in Section 2.3. A trained autoencoder may be used to separate waveforms that share the general characteristics of its training set from those that do not. Thus, the network can be used to 'bootstrap' data selection tasks: given a large data set to classify, we need to only perform visual assessment of a subset, upon which a network can be trained. This can then be used to extend the operator's selection across the entire data set.

We demonstrate this concept by using the autoencoder to identify a 'visually good' subset amongst a surface wave data set. This task is similar to that considered in Valentine & Woodhouse (2010), and much of the discussion of the goals and pitfalls of automation within that paper will also apply here. As already stated, the autoencoder demonstrated in Section 4 was trained on a hand-picked set of 1000 high-quality surface wave seismograms. As a result, the data space $\mathbb{S}^N$ for this autoencoder approximates the set of all ideal 512-point surface wave seismograms, and the reconstruction error provides a measure of how well a given waveform fits the characteristics of this set.

A further 1000 seismograms—previously unseen by the network but conforming to the selection criteria used for the training set (see Section 4)—are encoded and reconstructed. Some examples of waveforms selected at random from this set are shown in Fig. 16, along with their reconstructions and associated numerical error. As expected, we see that all reconstructions lie within $\mathbb{S}'^N$, and have the general characteristics of surface wave seismograms; furthermore, we see that the reconstruction error correlates well with the 'visual quality' of the original trace.

A basic selection algorithm to exploit this involves defining some threshold reconstruction error, E_{thresh}, used to determine whether a trace is sufficiently similar to $\mathbb{S}^N$ to be retained. To assess the performance of such an approach, we visually assess each of the 1000 seismograms used here as 'good' or 'bad', without prior knowledge of the reconstruction error. We then plot trade-off curves, as shown in Fig. 17, depicting the percentage of 'visually bad' traces removed from the data set as E_{thresh} is increased, along with the percentage of desirable traces lost. We find that performance is reasonable: in this example, virtually all 'bad' traces can be removed at a cost of around 40 per cent of the good traces; alternatively, 90 per cent of good traces may be retained if we also accept that the data set may contain about 10 per cent of the 'visually bad' waveforms.

We have therefore succeeded in extrapolating our classification of one data set onto another. Of course, performance is not perfect, and further development would be required before any practical implementation of this concept: other measures of similarity might yield better results, and it would be important to develop tools for monitoring performance, to ensure that any material misclassifications are detected. However, this offers a powerful method for handling large data sets automatically. We emphasize that this method extends beyond simple quality assessment—the approach might be adaptable to isolate particular phase arrivals within a catalogue, for example.

Importantly, the assessment process is fast: with a trained network, processing rates of hundreds or thousands of waveforms per second are easily achieved using a standard workstation. However, generating a new autoencoder is relatively costly—for the networks presented in this paper, a training time of 1–2 hr is typical—with requirements scaling according to the size of training data set, and the total number of weights in the network. This overhead cost must be considered when assessing the viability of a network-based approach to classification: where the data set to be classified is small, automation may lead to an *increase* in the total time required. However, for handling large waveform databases, the time saving is likely to be substantial. An automated method is also well suited to problems where real-time classification is necessary, or where the ability to reproduce a selection process at some later date is desirable, perhaps as new data become available.

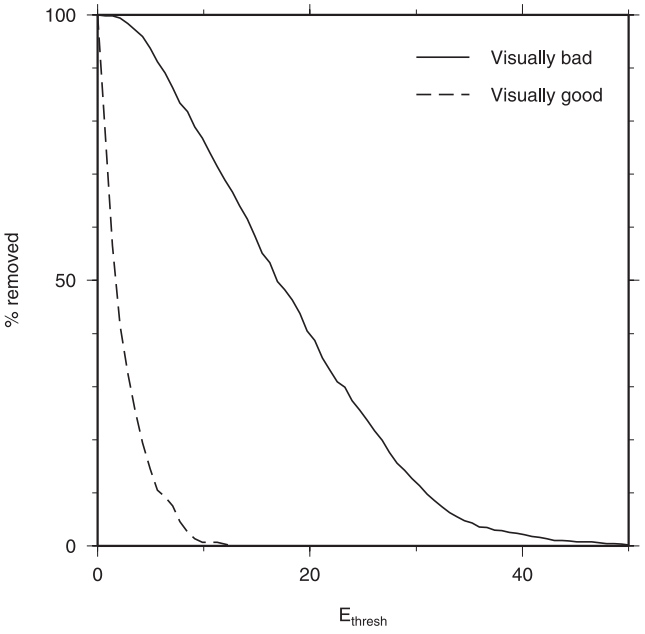

Figure 17. The autoencoder as a tool for quality assessment. An autoencoder was trained on a visually 'good' data set, and then applied to a new data set containing 1000 traces visually assessed as either 'good' or 'bad'. We adopt a strategy of discarding all traces for which the reconstruction error, E_i, exceeds a threshold E_{thresh}. As E_{thresh} is increased, we remove progressively more of the 'bad' traces, at the expense of some 'good' waveforms.

5.4 Concluding remarks

The applications suggested here are not exhaustive, and neither is our exploration of the properties of autoencoder networks. We do not suggest that the network architecture and training algorithms presented here are optimal; in any case, the definition of 'optimal' is likely to vary according to the precise problem being considered. Changes in network topology, the use of different neuron functions and alterations to the measure of reconstruction error that is minimized during training will all affect the performance of the autoencoder, and its ability to represent a given data set in fewer dimensions. However, our experience suggests that it is *not* difficult to obtain *satisfactory* results, and that the effects of such parameters are less significant than might be supposed. We believe that autoencoders show great promise for a wide range of geophysical applications—although our focus has been purely seismological, the method applies equally to any data set where apparent complexity may belie a simple representation. We therefore hope that it may allow greater insight into data sets, and thus into the Earth.

ACKNOWLEDGMENTS

We are grateful to the editor, Wolfgang Friederich and three anonymous reviewers for their comments and suggestions. APV is supported by QUEST, an Initial Training Network funded under the 'Marie Curie Actions' of the European Commission (grant 238007).

REFERENCES

Ackley, D., Hinton, G. & Sejnowski, T., 1985. A learning algorithm for Boltzmann machines, *Cogn. Sci.,* **9,** 147–169.

Basheer, I. & Hajmeer, M., 2000. Artificial neural networks: fundamentals, computing, design, and application, *J. Microbiol. Methods,* **43,** 3–31.

Bishop, C., 1995. *Neural Networks for Pattern Recognition,* Oxford University Press, Oxford.

Candès, E. & Wakin, B., 2008. An introduction to compressive sampling, *IEEE Signal Process. Mag.,* **25,** 21–30.

Chakraborty, A. & Okaya, D., 1995. Frequency-time decomposition of seismic data using wavelet-base methods, *Geophysics,* **60,** 1906–1916.

Chen, H. & Murray, A., 2003. Continuous restricted Boltzmann machine with an implementable training algorithm, *IEE Proc., Vis. Image Signal Process.,* **150,** 153–158.

Cheng, B. & Titterington, D., 1994. Neural networks: a review from a statistical perspective, *Stat. Sci.,* **9,** 2–30.

Cooley, J. & Tukey, J., 1965. An algorithm for the machine calculation of complex Fourier series, *Math Comput.,* **19,** 297–301.

Cottrell, G., 2006. New life for neural networks, *Science,* **313,** 454–455.

Dai, H. & MacBeth, C., 1995. Automatic picking of seismic arrivals in local earthquake data using an artificial neural network, *Geophys. J. Int.,* **120,** 758–774.

Diersen, S., Lee, E.-J., Spears, D., Chen, P. & Wang, L., 2011. Classification of seismic windows using artificial neural networks, *Proc. Comp. Sci.,* **4,** 1572–1581.

Donoho, D., 2006. Compressed sensing, *IEEE Trans. Inf. Theory,* **52,** 1289–1306.

Gentili, S. & Micheli, A., 2006. Automatic picking of *P* and *S* phases using a neural tree, *J. Seismol.,* **10,** 39–63.

Herrmann, F., Erlangga, Y. & Lin, T., 2009. Compressive simultaneous full-waveform simulation, *Geophysics,* **74,** A35–A40.

Hinton, G., 2002. Training products of experts by minimizing contrastive divergence, *Neural Comput.,* **14,** 1771–1800.

Hinton, G., 2010. A practical guide to training Restricted Boltzmann Machines, Tech. Rep. 2010-003, Department of Computer Science, University of Toronto.

Hinton, G. & Salakhutdinov, R., 2006. Reducing the dimensionality of data with neural networks, *Science,* **313,** 504–507.

Ho, T., 2009. 3-D inversion of borehole-to-surface electrical data using a back-propagation neural network, *J. appl. Geophys.,* **68,** 489–499.

Kohonen, T., 1988. An introduction to neural computing, *Neural Netw.,* **1,** 3–16.

Mackay, D., 2003. *Information Theory, Inference and Learning Algorithms,* Cambridge University Press, Cambridge.

Maggi, A., Tape, C., Chen, M., Chao, D. & Tromp, J., 2009. An automated time-window selection algorithm for seismic tomography, *Geophys. J. Int.,* **178,** 257–281.

Mas, J. & Flores, J., 2008. The application of artificial neural networks to the analysis of remotely sensed data, *Int. J. Remote Sensing,* **29,** 617–664.

Meier, U., Curtis, A. & Trampert, J., 2007. Global crustal thickness from neural network inversion of surface wave data, *Geophys. J. Int.,* **169,** 706–722.

Nyquist, H., 1928. Certain topics in telegraph transmission theory, *Trans. Am. Inst. Electr. Eng.,* **47,** 617–644.

Operto, S., Virieux, J., Hustedt, B. & Malfanti, F., 2002. Adaptive wavelet-based finite-difference modelling of SH-wave propagation, *Geophys. J. Int.,* **148,** 476–498.

Parker, D., Legg, T. & Folland, C., 1992. A new central England temperature series, 1772–1991, *Int. J. Climatol.,* **12,** 317–342.

Sandham, W. & Leggett, M., 2004. *Geophysical Applications of Artificial Neural Networks and Fuzzy Logic,* Kluwer Academic Publishers, Dordecht.

Shannon, C.E., 1948. A mathematical theory of communication, *Bell Syst. Tech. J.,* **27,** 379–423.

Shannon, C.E., 1949. Communication in the presence of noise, *Proc. Inst. Radio Eng.,* **37,** 10–21.

Shimshoni, Y. & Intrator, N., 1998. Classification of seismic signals by integrating ensembles of neural networks, *IEEE Trans. Signal Process.,* **46,** 1194–1201.

Simons, F. *et al.*, 2011. Solving or resolving global tomographic models with spherical wavelets, and the scale and sparsity of seismic heterogeneity, *Geophys. J. Int.,* **187,** 969–988.

Snieder, R., 1998. The role of nonlinearity in inverse problems, *Inverse Probl.,* **14,** 387–404.

Tingdahl, K. & de Rooij, M., 2005. Semi-automatic detection of faults in 3-D seismic data, *Geophys. Prospect.,* **53,** 533–542.

Tsaig, Y. & Donoho, D., 2006. Extensions of compressed sensing, *Signal Process.,* **86,** 549–571.

Valentine, A. & Woodhouse, J., 2010. Approaches to automated data selection for global seismic tomography, *Geophys. J. Int.,* **182,** 1001–1012.

van der Baan, M. & Jutten, C., 2000. Neural networks in geophysical applications, *Geophysics,* **65,** 1032–1047.

Wang, Y., Cao, J. & Yang, C., 2011. Recovery of seismic wavefields based on compressive sensing by an l_1-norm constrained trust region method and the piecewise random subsampling, *Geophys. J. Int.,* **187,** 199–213.

Ziv, J. & Lempel, A., 1977. A universal algorithm for sequential data compression, *IEEE Trans. Inf. Theory,* **23,** 337–343.

APPENDIX A: THE AUTOENCODER TRAINING ALGORITHM

Here, we derive the network training rules, as set out in eqs (21)–(23). The back-propagation algorithm is discussed in detail in most textbooks covering neural networks—see, for example, Bishop (1995). According to eq. (14), the layer-by-layer operation of the network can be described by

$$\mathbf{x}^{(n)} = \mathbf{f}\left(\mathbf{a}^{(n)} * \mathbf{b}^{(n)} + \mathbf{a}^{(n)} * \left(\mathbf{W}^{(n)}\mathbf{x}^{(n-1)}\right)\right), \tag{A1}$$

or, equivalently

$$x_{ij}^{(n)} = f\left(a_i^{(n)} b_i^{(n)} + a_i^{(n)} \sum_k W_{ik}^{(n)} x_{kj}^{(n-1)}\right), \tag{A2}$$

where the summation index k runs over the $K^{(n-1)}$ neurons in the $(n-1)$th layer and the activation function, $f(x)$, is as defined in eq. (12). We can therefore compute a variety of partial derivatives, viz.

$$\frac{\partial x_{ij}^{(n)}}{\partial b_\lambda^{(n)}} = \frac{1}{f_1 - f_0} a_i^{(n)} \left(x_{ij}^{(n)} - f_0\right)\left(f_1 - x_{ij}^{(n)}\right)\delta_{i\lambda}, \tag{A3}$$

$$\frac{\partial x_{ij}^{(n)}}{\partial W_{\lambda\mu}^{(n)}} = \frac{1}{f_1 - f_0} a_i^{(n)} x_{\mu j}^{(n-1)} \left(x_{ij}^{(n)} - f_0\right)\left(f_1 - x_{ij}^{(n)}\right)\delta_{i\lambda}, \tag{A4}$$

$$\frac{\partial x_{ij}^{(n)}}{\partial a_\lambda^{(n)}} = \frac{1}{f_1 - f_0} \left(x_{ij}^{(n)} - f_0\right)\left(f_1 - x_{ij}^{(n)}\right) \cdot \left(b_i^{(n)} + \sum_k W_{ik}^{(n)} x_{kj}^{(n-1)}\right)\delta_{i\lambda}, \tag{A5}$$

$$\frac{\partial x_{ij}^{(n)}}{\partial x_{\lambda\mu}^{(n-1)}} = \frac{1}{f_1 - f_0} \left(x_{ij}^{(n)} - f_0\right)\left(f_1 - x_{ij}^{(n)}\right) a_i^{(n)} W_{i\lambda}^{(n)} \delta_{j\mu}, \tag{A6}$$

with δ_{ij} representing the Kronecker Delta.

We wish to minimize the error function

$$E = \frac{1}{2Q} \sum_{i,j} \left(x_{ij}^{(L)} - x_{ij}^{(0)}\right)^2, \tag{A7}$$

as defined in eq. (17), by adjusting the various $a_i^{(n)}$, $b_i^{(n)}$ and $W_{ij}^{(n)}$ parameters. We therefore seek the partial derivatives of E with respect to these. Because the network treats each example in the data set independently, the chain rule for partial derivatives allows us to write

$$\frac{\partial x_{ij}^{(n)}}{\partial x_{\mu\nu}^{(n-m)}} = \sum_{\alpha,\beta,\ldots,\lambda} \frac{\partial x_{ij}^{(n)}}{\partial x_{\alpha\nu}^{(n-1)}} \frac{\partial x_{\alpha\nu}^{(n-1)}}{\partial x_{\beta\nu}^{(n-2)}} \cdots \frac{\partial x_{\lambda\nu}^{(n-m+1)}}{\partial x_{\mu\nu}^{(n-m)}} \delta_{j\nu}\,, \tag{A8}$$

with summation running over all neurons in the appropriate layer. This leads us to

$$\frac{\partial E}{\partial b_\mu^{(n)}} = \frac{1}{Q} \sum_{j,\alpha,\beta,\ldots,\lambda} \left(x_{\alpha j}^{(L)} - x_{\alpha j}^{(0)}\right) \frac{\partial x_{\alpha j}^{(L)}}{\partial x_{\beta j}^{(L-1)}} \cdots \frac{\partial x_{\kappa j}^{(n+1)}}{\partial x_{\lambda j}^{(n)}} \frac{\partial x_{\lambda j}^{(n)}}{\partial b_\mu^{(n)}}\,. \tag{A9}$$

Expressions for the partial derivatives with respect to $a_\mu^{(n)}$ and $W_{\lambda\mu}^{(n)}$ are similar.

In order to maximize computational efficiency, we wish to cast these equation in terms of matrix operations. As in eq. (20), we define the matrix $\mathbf{u}^{(n)}$ as having elements

$$u_{ij}^{(n)} = \frac{1}{f_1 - f_0}\left(x_{ij}^{(n)} - f_0\right)\left(f_1 - x_{ij}^{(n)}\right). \tag{A10}$$

This allows us to rewrite eq. (A6) as

$$\frac{\partial x_{ij}^{(n)}}{\partial x_{\lambda\mu}^{(n-1)}} = u_{ij}^{(n)} a_i^{(n)} W_{i\lambda}^{(n)} \delta_{j\mu}\,. \tag{A11}$$

Based on eq. (A9), we also define the quantity $\mathbf{\Delta}^{(n)}$

$$\begin{aligned}\Delta_{ij}^{(n-1)} &= \sum_k \Delta_{kj}^{(n)} \frac{\partial x_{kj}^{(n)}}{\partial x_{ij}^{(n-1)}} \\ &= \sum_{kj} \Delta_{kj}^{(n)} u_{kj}^{(n)} a_k^{(n)} W_{ki}^{(n)}\,,\end{aligned} \tag{A12}$$

$$\Delta_{ij}^{(L)} = x_{ij}^{(L)} - x_{ij}^{(0)}\,; \tag{A13}$$

or, in matrix form,

$$\mathbf{\Delta}^{(n-1)} = \mathbf{W}^{(n)\mathbf{T}}\left(\mathbf{\Delta}^{(n)} * \mathbf{u}^{(n)} * \mathbf{a}^{(n)}\right) \tag{A14}$$

$$\mathbf{\Delta}^{(L)} = \mathbf{x}^{(L)} - \mathbf{x}^{(0)}\,, \tag{A15}$$

as given in eqs (18)–(19). These terms may be interpreted as the difference between network outputs and inputs being 'back-propagated' through the network, to provide a measure of the effective error in the values computed for intermediate layers. We can now write

$$\frac{\partial E}{\partial b_\mu^{(n)}} = \frac{1}{Q} \sum_{i,j} \Delta_{ij}^{(n)} \frac{\partial x_{ij}^{(n)}}{\partial b_\mu^{(n)}} \tag{A16}$$

$$= \frac{1}{Q} \sum_j \Delta_{\mu j}^{(n)} u_{\mu j}^{(n)} a_\mu^{(n)}\,, \tag{A17}$$

where we have made use of eq. (A5). Similar expressions may be derived for the partial derivatives with respect to $a_\mu^{(n)}$ and $W_{\lambda\mu}^{(n)}$.

We therefore find that the overall error in the network, eq. (17), should be reduced by making the updates

$$b_i^{(n)} \to b_i^{(n)} - \frac{\eta}{Q} \sum_j \Delta_{ij}^{(n)} u_{ij}^{(n)} a_i^{(n)} \tag{A18}$$

$$W_{ij}^{(n)} \to W_{ij}^{(n)} - \frac{\eta}{Q} \sum_k \Delta_{ik}^{(n)} a_i^{(n)} u_{ik}^{(n)} x_{jk}^{(n-1)} \tag{A19}$$

$$a_i^{(n)} \to a_i^{(n)} - \frac{\eta}{Q} \sum_j \Delta_{ij}^{(n)} u_{ij}^{(n)} \left(b_i^{(n)} + \sum_k W_{ik}^{(n)} x_{kj}^{(n-1)}\right), \tag{A20}$$

as given in eqs (21)–(23). Note that all updates are made simultaneously—that is, all updates are computed using the values of parameters *prior* to the update stage. Once the update has been made, the network calculates new values for the $\mathbf{x}^{(n)}$ according to eq. (14), and the entire process is repeated until convergence is reached (as described in the main text).The 'learning rate', η is a small positive constant: clearly, there is a trade-off between the stability of the training algorithm, and the amount of time taken to reach a solution.