Adaptive subtraction using a convolutional neural network

Amarjeet Kumar¹, Gary Hampson¹ and Tom Rayment¹.

Abstract
Adaptive subtraction is a well-established routine technique in seismic data processing that relies on being able to match, in some sense, one dataset to another. It is a necessary part of many processing steps such as de-noising and de-multiple. The same matching methods are also used for advanced imaging steps such as non-stationary estimation of the Hessian as part of image-domain least-squares migration. Predictions of noise are often not as accurate as we would like because there are often shortcomings in the prediction method and because pragmatic acquisition compromises are made. The ubiquity of adaptive subtraction in the seismic processing workflow means there is always motivation to improve its performance. In this work, we propose a deep-learning approach for the adaptive subtraction of predicted multiple models from the input recorded data. It turns out that it is straightforward to use deep-learning networks to perform adaptive subtraction and as a result, benefit from the power of complex non-linear filters. Inspired by the idea of a convolutional autoencoder, we have developed a supervised convolutional autoencoder for adaptive subtraction. We demonstrate the proposed approach on synthetic and field data examples and compare the results using conventional adaptive subtraction methods to illustrate the effectiveness of our approach.

Terminology
This work reminded us that terminology can be as much a barrier, as an enabler, to discovery. We have, by necessity, used many terms from the field of machine learning. Since these terms are unfamiliar to many of us, we have italicised them in the text and collected them together in a brief glossary provided in the Appendix.

Introduction
Predictive multiple suppression methods consist of two main steps: prediction and subtraction. In the prediction step, models of multiples are estimated using either data-driven methods, such as SRME (Verschuur et al., 1992), or model-driven methods like deterministic water layer multiple modelling (Berryhill and Kim, 1986). In the subtraction step, these predicted multiples are modified before subtraction because they do not accurately match the real multiples in the observed data. This adaption of the modelled multiples is a very important step that ideally leads to perfect subtraction while preserving the primary events. There are numerous reasons why multiple model estimates are inaccurate including incomplete sampling of the wavefield (acquisition geometry, 2D versus 3D) and inherent limitations of the prediction algorithms. Predicted multiples may include wavelets and directivity patterns that differ from those present in the input data (Ikelle et al., 1997). In SRME, imperfections are commonly due to incomplete acquisition sampling (Dragoset and Jericevic, 1998).

The most common adaptive subtraction method is known as least-squares adaptive subtraction which is performed in the time-space domain. It locally minimizes the energy in the difference between the observed data and the filtered multiple model(s) by adjusting the filter coefficients. It works well if the primaries satisfy the minimum energy condition and they are orthogonal to the multiples (Wang, 2003). One way to avoid the minimum energy assumption is to use an L1-norm based matching filter (Guitton and Verschuur, 2004). Adaptive subtraction in other domains (e.g., Radon, wavelet, curvelet, seislet etc.) has been explored, each domain having associated advantages and disadvantages. The complex curvelet domain (Hermann et al., 2008) is particularly attractive because the seismic data is transformed into amplitude and phase as a function of scale, location and orientation, providing more possibilities for discrimination. A number of authors followed the work of Hermann et al. (2008), exploring curvelet domain adaptive subtraction methods to overcome the limitations of the time-space domain approaches (Saab et al., 2007; Neelamani et al., 2010; Wu and Hung, 2015; Kumar et al., 2019).

Despite these advances, there continues to be a desire for improved methods of subtracting multiples from the recorded data. Recently there has been a rapidly growing interest in the use of machine learning (ML) in the seismic data processing community. In the past few years, a growing number of researchers have attempted to use deep-learning (DL) algorithms to address problems such as trace-interpolation, de-noising, de-multiple and de-blending.

¹ DownUnder GeoSolutions
* Corresponding author, E-mail: amarjeetk@dug.com
DOI: 10.3997/1365-2397.fb2021066
(e.g., Mikhailik and Faul, 2018; Siahkoohi et al., 2018a, Garg et al., 2019; Mandelli et al., 2019; Yu et al., 2019; Zhang et al., 2019; Sun et al., 2020; Qu et al., 2021). DL is a subset of ML techniques in which layers of artificial neural networks are ‘trained’ to recognize patterns by exposure to large quantities of data examples. Although DL methods are computationally intensive during the training process, once the network is sufficiently well trained, application of what has been learnt can be orders of magnitude faster.

In this paper, we propose an alternative adaptive subtraction method which we base on a DL approach. This choice is motivated by the similarities between the conventional least-squares adaptive subtraction method and DL methods. In convolutional DL approaches, there are many hidden layers (all layers except the first and last), of which at least one is convolutional, that is, it contains an array of convolutional filters. Each of those filters is applied to the outputs of the previous layer. The output of each filter is passed through a non-linear activation function (a thresholding function). In the case that there is only one layer and the activation functions are removed then the DL network reduces to a conventional adaptive subtraction approach. Therefore, it is easily seen that DL can be considered as a complex non-linear system to design filters that match one data set to another.

We will begin by investigating a particular convolutional neural network (CNN) referred to as a convolutional autoencoder (CAE) (Goodfellow et al. 2016) and then introduce a modification which we refer to as a supervised convolutional autoencoder (SCAE). Using a SCAE we evaluate the performance of two loss (objective) functions (L2 and L1) on synthetic and field data examples. Finally, we compare our results to a typical least-squares adaptive subtraction method to illustrate the effectiveness of this new approach.

Method

Least-squares adaptive subtraction

Let \( d \) denote the recorded seismic data in the time-space domain and \( m_k \) represent the \( k \)-th of \( N \) estimated multiples derived by various means such as SRME, Radon domain estimates or shallow water multiple model (Kumar et al., 2017). Let the least-squares objective function for adaptively subtracting \( m_k \) from \( d \) be written,

\[
J = \| d - \sum_{k=1}^{N} f_k * m_k \|_2,
\]

where \( f_k \) is a single or multichannel matching filter in the time-space domain. In this approach, if the ensemble of filtered multiples is orthogonal to the primaries, the minimum of the objective function corresponds with complete multiple removal and perfect primary preservation. In this work, we replace this matching filter, \( f_k \), with a type of CAE. We chose a CAE because it can learn compact or sparse representations of the data without supervision.

Convolutional Autoencoder

A convolutional autoencoder (CAE) is an unsupervised learning algorithm that is trained to adjust its weights so that the output approximates the input based upon a sparse representation (latent representation). Figure 1 shows a simple CAE (Theis et al., 2017) which is a DL model composed of an encoder, a decoder and a bottleneck layer. The encoder maps the input, \( x \), to a lower dimensional, latent representation, \( h \), in the bottleneck layer. The decoder reconstructs the estimated image, \( \hat{x} \), from its latent representation, \( h \), through the inverse mapping. For illustration, we may write equations for a basic encoder and decoder in which there is only one hidden layer. The \( n \)-th feature map in the latent representation is \( h^i = \Psi(W^i * x + b^i) \), in which \( W^i \) represents the filters, \( b^i \) is the bias for the \( n \)-th feature map and \( \Psi \) is a nonlinear activation function (e.g., ReLU, ELU, LeakyReLU etc.). The decoder’s reconstructed version of the input is

\[
\hat{x} = \hat{\Psi}\left(\sum_{i=0}^{N} W^{i'} * h^i + c\right)
\]

in which \( W^{i'} \) is the adjoint of \( W^i \), \( c \) is the bias and \( H \) is the set of feature maps in the latent representation. The parameters \( W^i, W^{i'}, b^i \) and \( c \) are called the model parameters of the network and are determined by minimizing the loss function (objective function) using a sufficiently large number of training examples. CAEs are primarily used as a powerful tool for dimension reduction and image compression (Schiavon et al., 2019). Therefore, it is easy to understand this as a type of sparse representation or rank reduction.

Supervised convolutional autoencoder

The CAE is an unsupervised learning model; given \( x \), find the network model parameters to produce optimum estimates of the input, \( \hat{x} \). However, we can modify the CAE by asking it to find model parameters to convert the input, \( x \), to a different output, \( y \). We choose to refer to this as a supervised CAE (SCAE). This is similar to the U-net architecture of Ronneberger et al. (2015) with the skip connections removed. U-Net is a convolutional network originally developed for the classification of medical images. In our case, we do not wish to map \( x \) to \( y \), rather, we would like \( x \) to match part of \( y \). Therefore, we train a network to produce estimates, \( \hat{x} \), of the input, \( x \), that minimizes the misfit between \( x \) and \( y \) in a least-squares (L2-norm) or least-absolute (L1-norm)
Network architecture

Just as the design of adaptive subtraction workflows substantially influences the quality of the results, so too does the design of a network’s architecture for particular applications. This network design problem is a non-linear optimization and has many parameters. Our design proceeded by experimentation, reading around the subject and recognizing that the non-linear nature of the problem meant that we would, at best, end up in a useful local, rather than global, minimum. After experimenting with the number of layers in the encoder and decoder, we found that four encoder layers and four decoder layers seemed to perform the best. The output of each layer passes through an activation function before passing to the next layer. It is these activation functions (types of thresholding functions) that can impose non-linear characteristics on the network. The encoder performs 2D convolutions in each layer, with the number of channels/feature maps in each layer typically reducing from layer to layer. In contrast, the decoder performs transpose convolutions in each layer with the number of channels typically increasing from layer to layer.

The middle layer, known as the bottleneck, contains the most compressed representation of the features that have been learnt (often called latent representation). The latent representation is heuristically rather like a dictionary that permits a sparse representation of the training data. Our final network is shown in Figure 3. It has a total of nine layers (including the bottleneck) with the number of feature maps shown in each layer. The input and output layers reflect the patch sizes (64x64 samples) and the latent representation in the bottleneck has 14x14 samples per feature map. We found that the best results were obtained using a convolutional filter size of 4x4. The first and last layers do not sense. We refer to it as a supervised convolutional autoencoder (SCAE) which is shown in Figure 2. In this context, $x$ represents the predicted multiple model(s) and $y$ represents the recorded input data (primaries + multiples). The SCAE network is trained over sufficiently many training pairs $(x_i, y_i)$ to optimize the model parameters capable of performing adaptive subtraction. To emphasize the similarity with conventional adaptive subtraction that was described earlier, we may write the new objective function for the SCAE filters by replacing the matching filter $f_k$ with the SCAE acting on the multiple model, $m$, in equation 1,

$$J = \| d - \text{SCAE}(m) \|^2.$$

(2)

Training strategy

Neural networks learn more quickly if the inputs ($x$) and their target outputs ($y$) have similar amplitudes. Therefore, it is very common to perform standardization on the data. This process subtracts the mean value and scales the standard deviation to unity. It is applied to the input and the output data. We applied standardization in our experiments because the amplitudes of predicted multiple models are often significantly different to those of the recorded data.

For training, we randomly selected a subset of shot-gathers from the input dataset and divided each shot into a number of regular overlapping patches based on patch size and patch separation. This is equivalent to overlapping window-based seismic data processing. With $N$ training shot-gathers and $P$ patches per shot, we have a total of $NP$ patches to train the network. These patches were then randomly shuffled and fed into the network for training. This random ordering is intended to avoid biases developing during the training.

Figure 2 A flow diagram of the proposed supervised convolutional autoencoder. Here, the output $Y$ is different to the input, $X$. The network learns to estimate data without multiple contamination by minimizing the loss function.

Figure 3 The network architecture of our supervised convolutional autoencoder (SCAE).
number of samples in each feature map is controlled by the stride (decimation of filter output) of the filter, zero-padding, pooling (represent a group of pixels as the maximum or average to reduce the number of samples) and up-sampling (a range of interpolation methods to create more samples). The output channels of each layer were passed through a non-linear activation function called an exponential linear unit (ELU).

**Sigsbee Data Example**

We demonstrate the application of the proposed approach applied to a synthetic dataset and a field data example. The synthetic data were generated using the Sigsbee model (SMAART JV, 2001) and a 2D SRME process was performed to compute the predicted surface-related multiple models. For this test we built a dataset of 354 shot gathers, each with 320 traces and 768 time samples. We randomly selected 200 of those shots in pairs from the synthetic and SRME datasets for training. We tiled each of these shots into 207 overlapping patches each measuring 64x64 samples and overlapping by 50% in both directions. Some areas in the predicted models had no energy present, for example before the first order surface-related multiples. Patches in these areas were excluded from training because they did not
contribute beneficially to the learning, indeed, we saw a deterioration in the learning process. This selection process resulted in a total of approximately 40,000 training pairs \((X,Y)\) as input to the network. An example of some of the training pairs is shown in Figure 4. Note from here on in each figure, the axes are all in sample and trace numbers unless stated otherwise. This is for the reader’s convenience to follow the numbers in the text.

We trained the network with a **batch size** of 64 (the number of patches used to derive 1 update) and a **learning rate** of 0.001 (gradient descent step size) for 50 **epochs** (number of complete passes through all the patches). The training was carried out on the Pytorch platform (Paszke et. al., 2017). Each **epoch** on a GPU node took approximately 25 s resulting in a total training time of about 20 minutes. After training we were able to perform adaptive subtraction in about 0.3 s per shot gather, which is comparable to conventional time-space least-squares adaptive subtraction.

We studied two **loss functions** (L2 and L1) and created a corresponding SCAE network for each. Figure 5 shows a normalized convergence graph of loss as a function of **epoch** for both loss functions. It shows they reduce at roughly similar rates although the L1 loss function reduces further.

Figures 6 and 7 show the results of applying the trained adaptive subtractors to some of the training data. The L2 (L1) loss function was used in Figure 6 (7) respectively. It is clear that the network learns to produce good estimates of the true multiple model. There are some remnants of multiple energy remaining in the L2 results that are indicated by the yellow arrows. The L1 function performs better overall, although an exception is indicated by the red arrow.

---

**Figure 7** Adaptive subtraction results using an L1 loss minimized SCAE network. Further suppression of indicated multiples are achieved using this network. There is slight deterioration of the result in the first patch (compared to results shown in Figure 6, indicated by a red arrow.)

**Figure 8** Adaptive subtraction results on a shot gather using the learnt L2 and L1 loss minimized SCAE network. A) Input shot gather from, b) results using L2 loss, c) results using L1 loss, and d) the difference between L2 and L1 loss results.
Figure 9: A comparison of results using the SCAE approach and a conventional least-squares adaptive subtraction. a) Input common offset gather, b) primaries using the L1 loss minimized SCAE network, and c) primaries using a conventional least-squares method. The red circles highlight the most complex region where conventional methods struggle.

Figure 10: A comparison between the true multiples and conventional least-squares adapted multiples. a) True multiples, b) Adapted multiples using a conventional least-squares method, and c) the difference plot showing the error between the true and the adapted multiples.

We then applied these networks on a subset of the data which had not been used for training. The results using both L2 and L1 loss networks are shown in Figure 8b and 8c, respectively. The difference between the L2 and L1 results is shown in Figure 8d. As we saw in the examples from the training dataset, although both approaches perform well, the L1 results typically show better multiple removal than L2.

We then applied the proposed SCAE to all of the shot gathers. For comparison, we also applied conventional least-squares adaptive subtraction to the same dataset (using a global match followed by a windowed match with windows the same size as the SCAE patches). Figure 9 shows a common offset section before adaptive subtraction (a), after L1 SCAE adaptive subtraction (b) and after conventional least-squares adaptive subtraction (c). The L1 SCAE adaptive subtraction has performed very well, removing much of the complex multiple that presents such a problem for conventional least-squares adaptive subtraction (particularly notable in the red circled area). The conventional approach has struggled to suppress the low frequency components of the multiples and a low frequency residue remains (compare...
Figures 9b and 9c). Using the modelled data without free-surface multiples, we compare the true multiples with our estimates of the adapted multiples and the error in that estimate. Figure 10 shows the true multiples (a), the adapted multiples from a conventional least-squares adaptive subtraction (b) and the difference between them (c). Similarly, Figure 11 shows the results for the SCAE adapted multiples. We quantified these results by calculating the signal-to-noise ratio (SNR) between the adapted multiples and the difference plot. The SNR of the conventional least-squares adaptive subtraction method is 18 dB compared to 22.4 dB for the SCAE-based approach. These results demonstrate that the SCAE adaptive subtractor shows significant potential for helping to deal with complex multiples.

**Field Data Example**

With encouragement from our synthetic data experiments, we tested the technique on a real field data example. There were 1158 input shot gathers taken from a sail-line in a 3D survey. The shots were selected from an inner 448 channel cable with 736 time samples. A 3D SRME had already been used to generate a multiple model derived using the full 3D dataset. We randomly select 350 of the shot gathers for training purposes. Each of these

---

**Figure 11** A comparison between the true multiples and the adapted multiples using the SCAE network. a) true multiples, b) Adapted multiples using the SCAE approach, and c) the difference plot showing the error between the true and the adapted multiples.

**Figure 12** Adaptive subtraction results on a test field data shot gather using the trained L2 and L1 loss SCAE network. a) input shot gather from the test datasets, b) result using L2 loss, c) result using L1 loss, and d) The difference between L2 and L1 loss results.
noticeable at the nearer offsets as well as in the critical multiples cone.

Figure 13 shows a common offset section for this dataset, before SRME (a), after L1 loss SCAE adaptive subtraction (b) and after a conventional least-squares based adaptive subtraction workflow (c). The red circle and the red arrow indicate areas where the L1 network is better than the conventional workflow. The yellow arrow shows an area where the conventional workflow is more effective than the L1 network.

Figure 14 shows a comparison of adapted multiples using the SCAE method and the production adaptive subtraction workflow. a) Input common offset section, b) Adapted multiples using the L1 SCAE network, and c) Adapted multiples using the conventional least-squares based workflow.

shots was tiled into 286 overlapping patches measuring 64x64 samples and overlapping by 50% in both directions. Patches, in which there was no predicted multiple information, were excluded from the training process. Approximately 61,000 training pairs remained for the training exercise.

The network was trained using a batch size of 64 and a learning rate of 0.001 for 100 epochs. For comparison, we trained an L2 and an L1 version of the network. The trained network was then applied to the shot gather data and an example of the results is shown in Figure 12. Both networks performed well; however, the multiples are slightly better suppressed using the L1 loss network. Figure 12d shows that the difference is noticeable at the nearer offsets as well as in the critical multiples cone.

Figure 13 shows a common offset section for this dataset, before SRME (a), after L1 loss SCAE adaptive subtraction (b) and after a conventional least-squares based adaptive subtraction workflow (c). The adapted multiples from both methods are also shown in Figure 14. The conventional workflow consisted of many steps, such as preconditioning of models, global matching, frequency splitting and local matching. It is a time-consuming art to design and optimize such workflows.

Overall, the L1 network is producing similar results to the conventional workflow, but there are important differences.
In the shallower zone, particularly the area highlighted by the red arrow and red circle, the L1 network has been much more effective in removing the water bottom multiple. In the deeper zone, indicated by the yellow arrow, the L1 network seems to have been less effective than the conventional workflow. Two possible conclusions can be drawn from this result: 1) Perhaps the SCAE approach should be part of a workflow in which other more conventional components also play a role. 2) The network may need additional information about non-stationary behaviour. For example, perhaps the network needs to know the depth of each training patch so that it can tailor its behaviour more closely with respect to depth. However, we leave those investigations for the future.

Conclusion and discussion
Adaptive subtraction is a very important step for the optimum subtraction of multiple models. It can often be a complex time-consuming task to achieve acceptable results that do not damage the primary signal. In this work, we have shown that a deep learning approach can be used to perform adaptive subtraction. We developed a supervised convolutional autoencoder network architecture that involved training the network using a random selection of shot gathers followed by the application of the trained network to the rest of the dataset. We compared the performance of the L1 and L2 loss functions. Our results indicate that typically the L1 network seems to perform better than the L2 network, although we observed exceptions that deserve further study. A possible approach might be to consider a linear combination of L1 and L2 loss functions as a trade-off solution.

Besides their non-linear power, a marked benefit of using neural networks is that once they have been trained, they perform their function extremely quickly. The question naturally arises: can we use our trained network on other datasets? Although it will work equally well on similar data from the same survey, we expect that the network would require completely retraining for other datasets. However, Shin et al. (2016) looked at the ability to use a pre-trained network for a transfer learning process. This would allow some modest degree of additional training to fine tune the model parameters instead of completely retraining the network.

In this article, we have limited our attention to a particular application of SCAE-based matching, as part of an adaptive subtraction process. However, we are happy to report that we have also been encouraged by applying this approach to a number of closely related problems, including estimating the non-stationary Hessian as part of image-domain least-squares migration. It seems, that as far as machine learning is concerned, there is no lack of candidates for smoother pebbles or prettier shells than ordinary, on the data processing beach.

Acknowledgements
We would like to express our appreciation to DownUnder GeoSolutions for allowing us to present this work and to our reviewers, Pierre Turquais and Anthony Day, for their helpful suggestions. In addition, we also thank editor-in-chief Peter Rowbotham for his valuable comments.

References
Convolutional autoencoder

A neural network that uses convolution with filters instead of simple scalar weighting.

Decoder

The part of an autoencoder after the bottleneck that reconstructs an estimate of the input data.

Deep learning

A neural network with a number of hidden layers.

Dictionary

A collection of essential features that can be combined to approximate the input.

Down-sample

Reduced number of samples in a feature map by pooling, convolution with stride>1 or not zero-padding before convolution.

ELU

Exponential linear unit — a type of activation function.

Encoder

The part of an autoencoder before the bottleneck that decomposes and compresses the input data into a sparse set of essential characteristic features.

Epoch

Exposure of every patch in the training dataset to the autoencoder once. Typically training requires many epochs.

Feature map

The result of applying one filter to the previous layer. Synonymous with channel (but possibly more descriptive).

Filter

A 2D or 3D linear convolutional filter.

Hidden layers

The layers between the input layer and the output layer. Parameters of the network set by the user, rather than by minimising the loss function.

Hyper-parameters

Parameters of the network set by the user.

Latent representation

The essential features contained in the bottleneck which can be combined by the decoder to produce an approximation of the input data.

Learning rate

This is equivalent to step-size in gradient descent algorithms.

Loss function

The objective or cost function of the network based on a norm (e.g., L1 or L2) of the misfit between the input and the reconstructed output.

Machine learning

A computer algorithm that adjusts its behaviour to better predict outcomes for a class of problems.

Model parameters

The weights, biases and filter coefficients derived from the data by loss function minimisation.

Patches

Data is presented to the network in patches (subsets of the input data) which are rectangular and of a specified patch size, e.g., 64x64.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pooling</td>
<td>The combining of samples under a stencil to produce one output sample. Used in down-sampling.</td>
</tr>
<tr>
<td>Supervised learning</td>
<td>Machine learning where the correct answer is provided to the network so that it can reduce its error.</td>
</tr>
<tr>
<td>Skip connections</td>
<td>Direct connections between layers in the encoder and decoder. Most usually to layers of the same dimension.</td>
</tr>
<tr>
<td>Transpose Convolution</td>
<td>2D or 3D correlation (really convolution because the filter is placed in reverse order). Beware, in some cases this is confusingly referred to as ‘deconvolution’.</td>
</tr>
<tr>
<td>Standardisation</td>
<td>To shift and scale a dataset so that it has zero mean and unit standard deviation.</td>
</tr>
<tr>
<td>Up-sample</td>
<td>To generate more samples in a feature map by various forms of interpolation.</td>
</tr>
<tr>
<td>Stride</td>
<td>The number of samples to shift a filter in each direction before calculating the next dot product of the convolution. One method of down-sampling.</td>
</tr>
<tr>
<td>Zero-padding</td>
<td>To extend one or more dimensions of a feature map with one or more zero values.</td>
</tr>
</tbody>
</table>