Synthetic seismic data generation with deep learning

G. Roncoroni [a,*], C. Fortini [b], L. Bortolussi [a], N. Bienati [b], M. Pipan [a]

[a] University of Trieste, Department of Mathematics and Geosciences, Trieste, Italy
[b] ENI – Upstream & Technical Services, San Donato Milanese, Italy

ARTICLE INFO

Article history:
Accepted 26 April 2021

Keywords:
Reflection seismic
Seismic modelling
Deep learning
Machine learning
Synthetic seismogram

ABSTRACT

We study the applicability of deep learning (DL) methods to generate acoustic synthetic data from 1D models of the subsurface.
We designed and implemented a Neural Network (NN) and we trained it to generate synthetic seismograms (common shot gathers) from 1-D velocity models on two different datasets: one obtained from published results and the other generated by Finite Differences (FD) numerical simulation. We furthermore compared the results from the proposed model with the published one.
Moreover, we tried to to add more flexibility to this methodology by allowing change of wavelet and the acquisition geometry. We obtained good results in terms of both computation efficiency and quality of prediction.
The main potentialities of the work are the low computational cost, a high prediction speed and the possibility to solve complex non-linear problems without knowing the physical law behind the phenomenon, which could led great advantages in the solution also of the inverse problem.
DL to generate 1-D acoustic synthetic seismograms without solving wave equation Solution to the 1-D problem through custom Recurrent Neural Network Retraining strategy to improve flexibility and applicability Computational complexity analysis.

1. Introduction

The main objective of this work is the implementation of Deep Learning (DL) solutions to generate synthetic seismograms from 1D acoustic models without solving the wave equation. This is done by training a NN model which after training is able to predict common shot gathers from 1-D velocity models.

The wave equation, is non linear with respect to velocity. Numerical solutions schemes, such as finite difference or pseudo-spectral schemes, are computationally demanding and the results may be affected by dispersion error and boundary reflections.

The NN approach is computational demanding only during NN training and is affected by the problems in the training dataset. The proposed methodology can solve the forward problem of seismic wave propagation faster than classical methods, especially when high-frequency source wavelets are considered.

Although direct application and computational advantages, being able to simulate data with a simple 2D geometry but taking into account the multiples reflection, leads the path to possible future works for a better and faster solution of the inverse problem.

The generation from text data of an audio track that simulates the human voice, is a similar problem that has been already tackled by DL approaches (see e.g. Oord, Dieleman, Zen, Simonyan, Vinyals, Graves, Kalchbrenner, Senior and Kavukcuoglu (2016)). The application of DL to sciences, and in particular to geophysics, is a new branch that has been rapidly developing in the last years benefiting to the big improvements in the cost-to-performance ratio of computational resources. The power of Deep Learning is due to the capacity of finding recurrent patterns or causality without the need of user judgment or explicit coding by capturing statistical relations to provide generative or discriminative models.

Oye and Dahl Øye and Dahl (2019) proposed a method based on Convolutive Neural Network (CNN) to estimate velocity models from Raw Shot Gathers, with possible applications to Full Waveform Inversion. This is a new approach that can overtake the computational limit of the old methods (such as genetic algorithms) used to estimate a velocity model. Also Yang and Ma Yang and Ma (2019) developed a technique to build seismic velocity models using U-net. Guo et al. Guo et al. (2019) used Bi-directional Long Short-Term Memory (LSTM) for seismic impedance inversion, an ill-posed and non-linear problem. The Bi-directional LSTM recurrent neural network applied to the inverse problem of P-impedance estimation can be attractive because it does not need a prior estimate of the wavelet, one of the main problems in this type of studies.

Moseley and Markham Moseley et al. (2019a) proposed a wavenet-like net Oord et al. (2016), a stack of causal CNN layers, for a fast approximate simulation of acoustic waves.

* Corresponding author.
E-mail address: groncoroni@units.it (G. Roncoroni).

Table 1
Parameters used to produce the second dataset.

Parameter	Set values
Sampling rate [s]	0.001 s
Listening time [s]	1 s
Grid point [x, y]	[600, 600]
Grid spacing [min, max]	[5, 5] m
Ricker peak *f* [Hz]	20 Hz
Velocity [min, max]	[1500 m/s, 5000 m/s]
Depth [min, max]	[100 m, 3000 m]
Number of layers	[3–8]

The methodology proposed in this work considers the seismogram as a time-series (as in speech generation) rather than a matrix of pixels (as in image processing). We have used a custom made Recurrent Neural Network Sherstinsky (2020) model to implement such strategy.

The result is a model able to learn how to solve the problem in 1-D-layered media and capable of an approach to increase flexibility and to reduce the computational cost in the training process.

The final goal is to speed up forward modelling for the wave equation with respect to the classical methodology and to provide a starting point for future applications to more complicated scenarios, e.g. elastic approximation, or to a new approach to the inverse problem.

2. Methods

2.1. Datasets

We used two different datasets to train NN and to evaluate the importance of the dataset on different training and prediction features: the first taken by Moseley et al. Moseley et al. (2019a), hereinafter

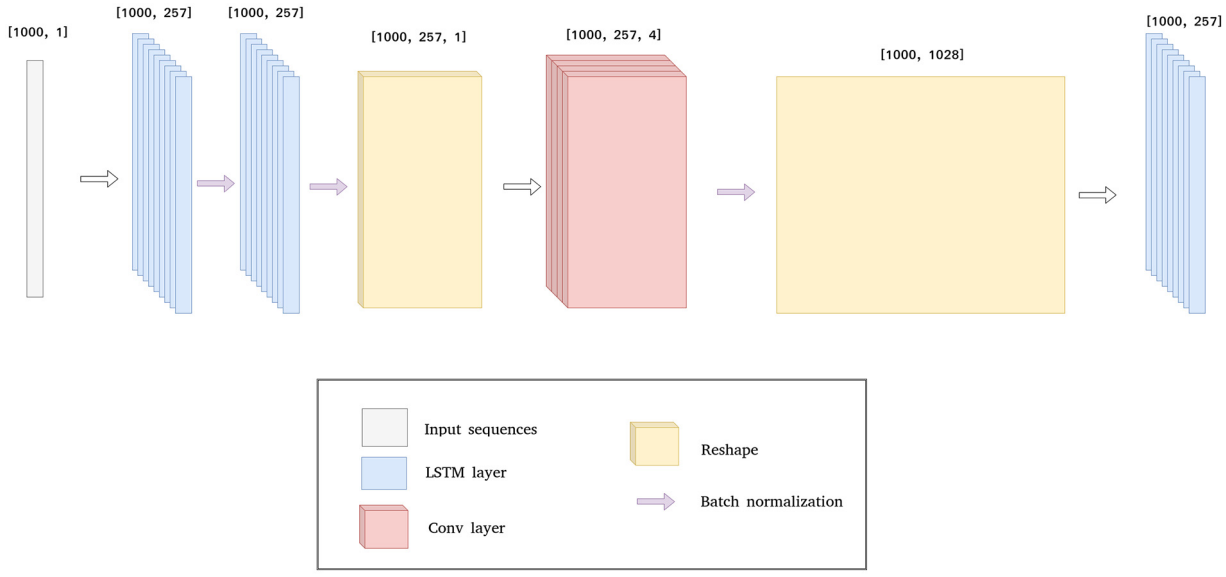

Fig. 1. The NN is made of two Long Short Term Memory (LSTM) layers, a Convolutive layer, and a final LSTM layer.

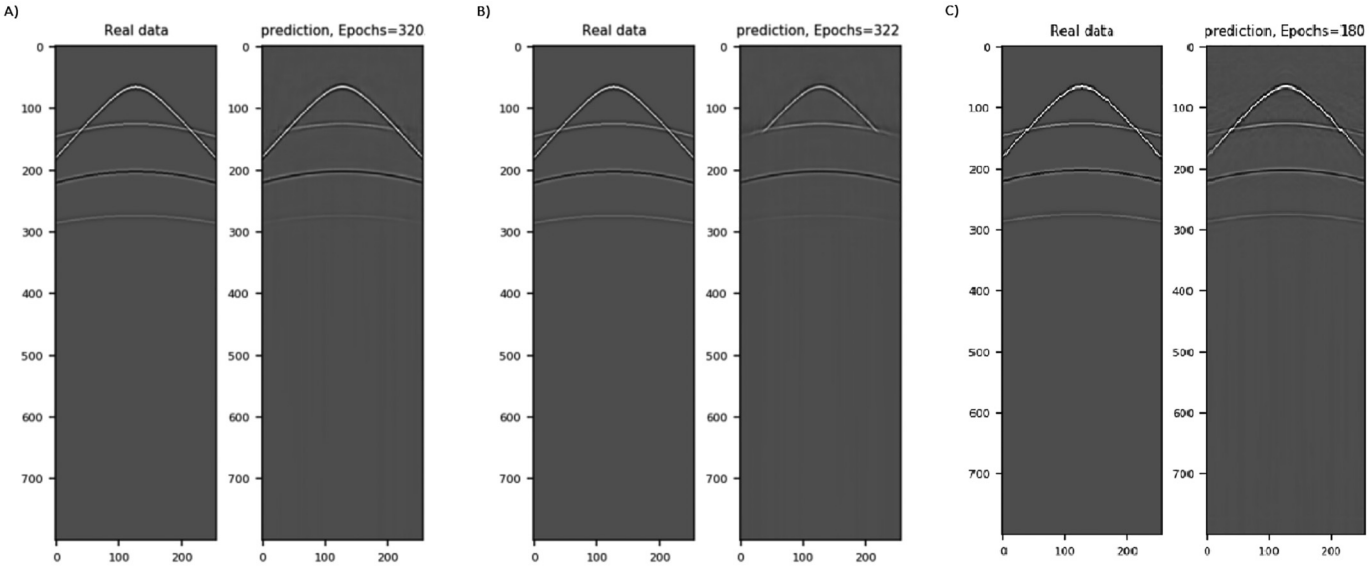

Fig. 2. Performance on overlapping signals during the training. Prediction without Convolutive Layer (A, B), at epochs 320 and 322 respectively, and with Convolutive Layer (C), at epochs 180.

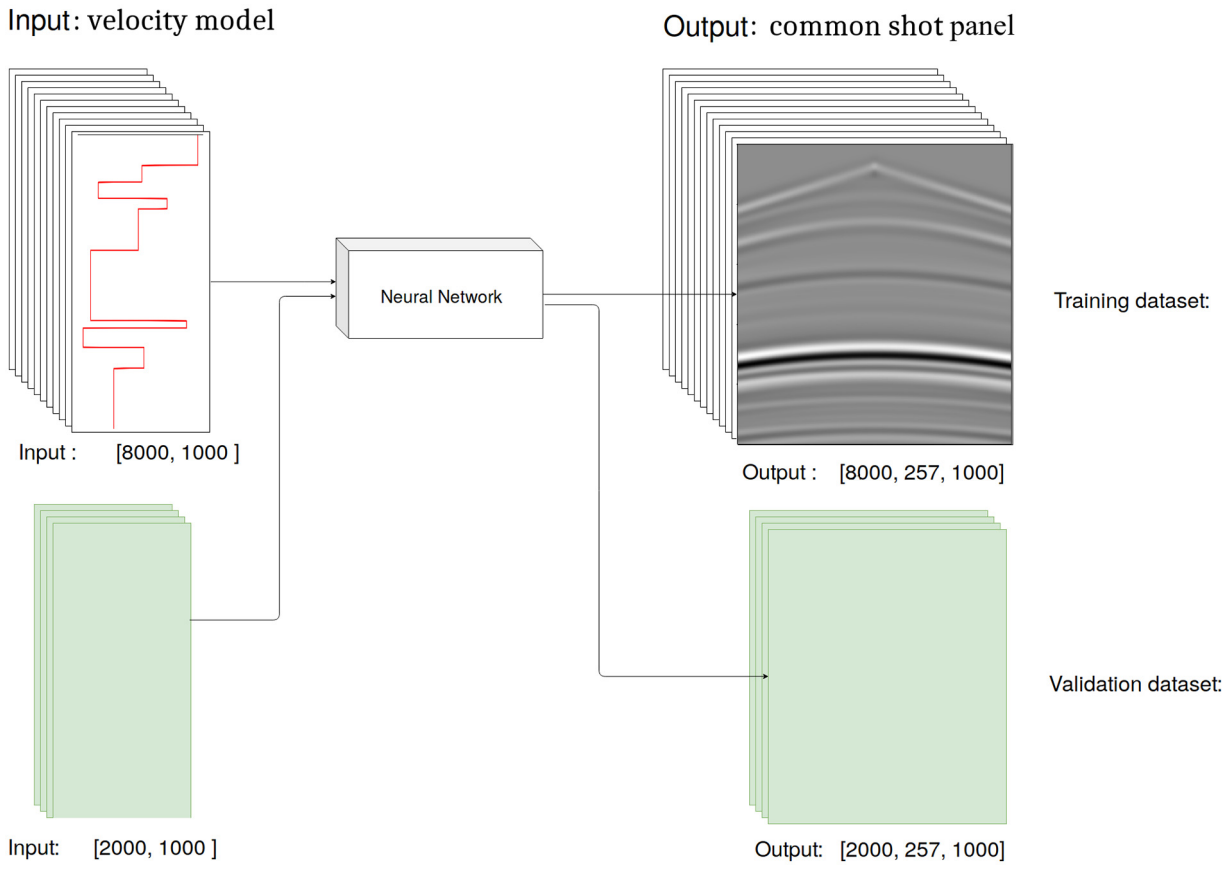

Fig. 3. The dataset is split in four tensors: two for the input and two for the output data (training and validation dataset, respectively). Input tensors are made of a number of velocity functions and output tensors of Common Shot Gather panels (CSG).

referred to as MOS, which was used to tackle the same task, the other generated by numerical simulation [with the codes provided in the Devito Project by Louboutin et al., 2019 Louboutin, Lange, Luporini, Kukreja, Witte, Herrmann, Velesko and Gorman (2019)], hereinafter referred to as DEV.

Both datasets consists of two tensors (training/validation): two for the input and two for the output data. The input tensor is made of a number of velocity functions (10,000) and the output tensor of Common Shot Gather panels (CSG; 10,000 panels of 11 seismic traces for MOS data and 257 seismic traces for DEV data).

We split both test datasets, made of 10,000 input velocity functions and 10,000 output CSG, in training and validation datasets (80% the former and 20% the latter).

We will hereafter use the notation tensor for multi-dimensional arrays: in particular, the output of the NN is a multi-dimensional array with size [10,000, 257, 1000].

MOS is available on GitHub Moseley et al. (2019b). It is made of a tensor with velocity functions and a tensor with the associated CSG panels. The 50% reduction in size of the dataset, compared to the original 20,000 sample Moseley's dataset, is due to the fact that our model is based on a reduced number of weights, compared to Moseley's one, and does not require large datasets. We have applied a random data selection and splitting procedure.

The CSG panel was computed by solving the wave equation by using finite differences over a 3Km x 3Km grid in a horizontally layered model. There are 11 evenly spaced receivers, with horizontal spacing of 200 m, and the source is at the 6th receiver position.

MOS and DEV mainly differ in number of receivers and source-receiver offset, i.e., DEV is the result of numerical simulation with the parameters shown in Table 1. We randomly generated different

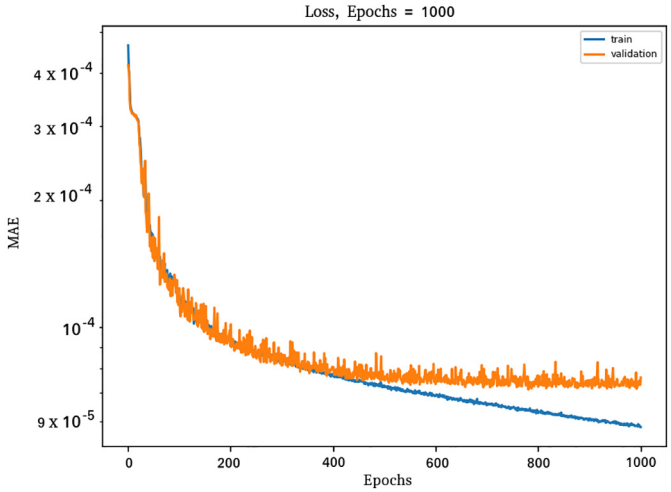

Fig. 4. Training of the proposed net on the MOS dataset: Loss vs Epochs plot. Orange line represents the trend of validation dataset prediction while blue line shows the training one.

subsurface models by using the velocity and depth constraints shown in Table 1.

In order to generate the input velocity function for FD (in depth) and for NN (in time) we used the following workflow:

for each velocity function:

n= generate a casual number of horizons.

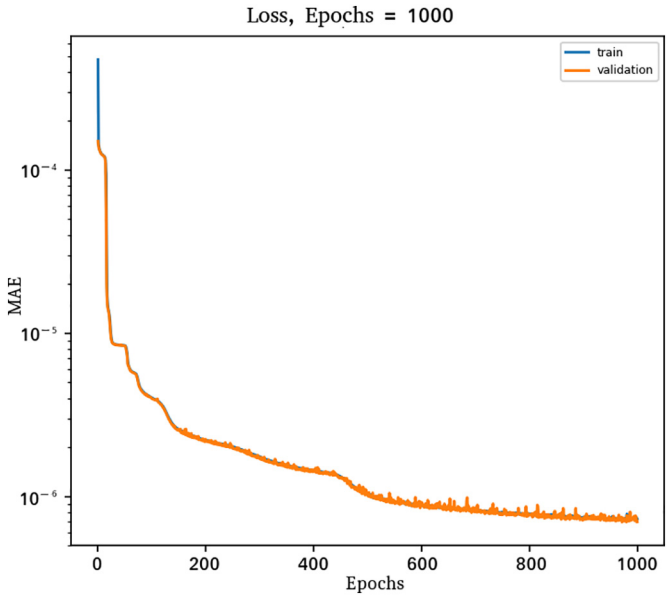

Fig. 5. Loss vs Epochs of the training on DEV dataset. Orange line represents the trend of validation dataset prediction while blue line shows the training one.

generate n random depth values.
sort depth values and append 0 at the beginning.
generate n + 1 velocity values.
build velocity function in depth.
convert depth to time.
build velocity function in time.
Normalize data between 0 (1000 m/s) and 1 (6000 m/s) ".

Data in DEV dataset has 257 evenly spaced receivers, with horizontal spacing of 5 m, and the source at the 129th receiver position.

2.2. NN architecture

We tested several architectures, varying the number of Hidden Layers (HL), neurons and with different number of filters and different kernel sizes in the convolutional layer. After an extensive tests phase, we were able to define the best performing architecture, shown in Fig. 1. This NN consists of two LSTM layers Hochreiter and Schmidhuber (1997), a convolutional layer Simard et al. (2003), and a final LSTM layer. We choose LSTM because they are recurrent NN, which means they can deal with causal tasks, and they have a long short-term memory, so they should better represent the physics behind the task.

All of the LSTM layers have a number of neurons equal to the number of sensors in the recording array to be simulated. The LSTM used in this model are implemented by the Keras layer CuDNNLSTM, a Fast LSTM implementation with NVIDIA CUDA Deep Neural Network library (cuDNN), a GPU-accelerated library of primitives for Deep Neural Networks (DNN) Chetlur et al. (2014). In the convolutional layer the kernel size is [64, 4] and it has 4 filters with a ReLu activation function Glorot et al. (2011). This gives the NN the flexibility to get values in the window [t - 4: t] from the 64 closer neurons, whit *t* the time steps, and to store it in 4 different filters.

The proposed DL architecture (Fig. 1) provides an output h at time t from three different input, namely: Xt, h(t-1) and the cell state line. The latter is from the method used to obtain the long term memory. The output is calculated from a series of weights, activation functions and simple mathematical operations. In case of overlapping signals, such as e.g. multiple/primary reflections, such architecture is unable to perform correct predictions at times larger than the overlap point. This is the reason for the introduction of the CNN layer: it increases the number of input to the same neuron thus allowing a correct prediction even in the post-overlap part of the record. Fig. 2 A,B,C illustrate the performance of such solution during the training: in Fig. 2 A,B (without Convolutive Layer) the loss function is stable from epoch 320 and the problem is not solved. Fig. 2 C (with Convolutive Layer) shows a successful prediction from epoch 180.

In order to train the NN, we need to minimize the norm of the difference between the prediction and the reference output, this is called loss function: the most used loss functions in regression tasks are the mean square error and the mean absolute error.

However, in our scenario, geometrical spreading and transmission losses cause an amplitude decay with time, such that the loss function is sensitive to the direct wave only due to its dominant amplitude and NN outputs zeros. Accordingly, in the first implementation, the NN

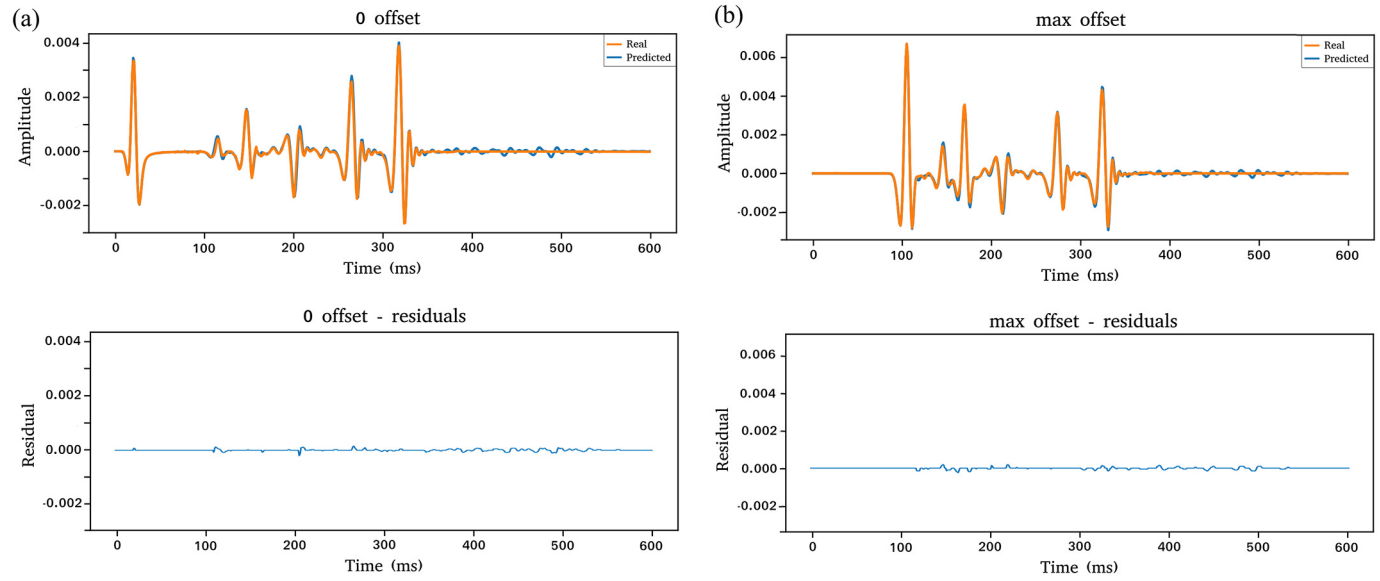

Fig. 6. Prediction and residuals of a predicted CSG panel. 0-offset prediction and residuals (a), max-offset prediction and residual (b). In red we highlighted the phase shift in the residuals.

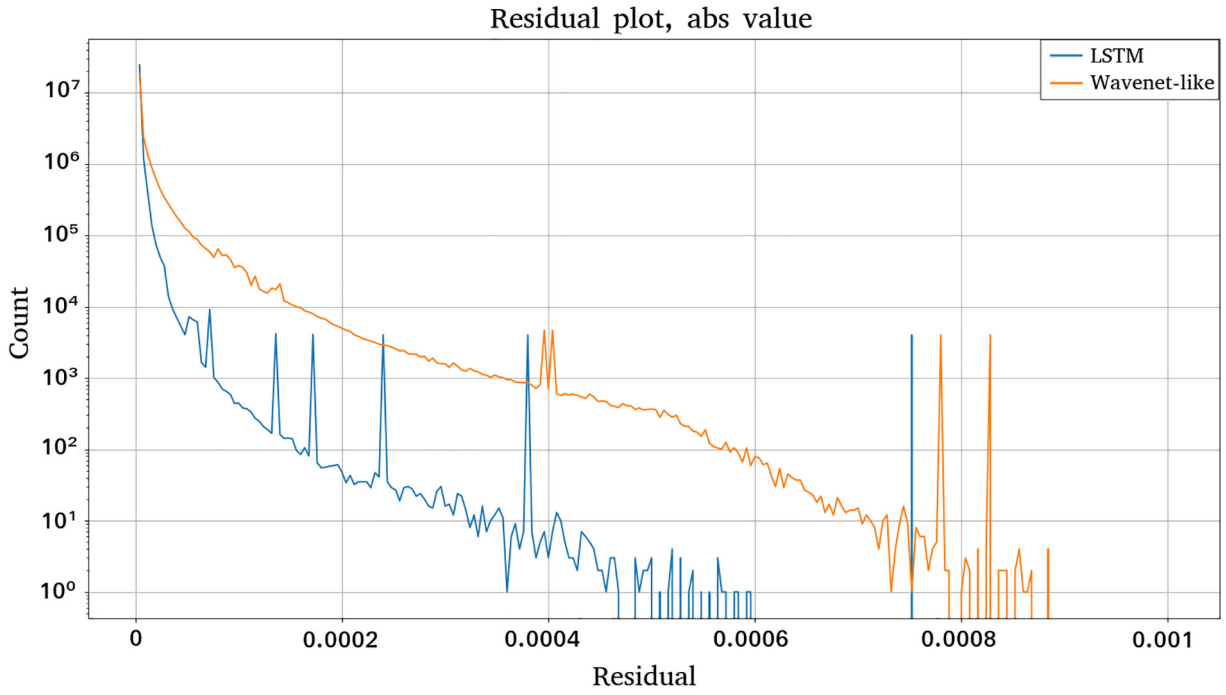

Fig. 7. Residual plot of both models: LSTM model in blue, WaveNet-like Oord et al. (2016) model in orange. Vertical axis is logarithmic. Horizontal axis, called residual, is made of 50 linearly divided bins in a range of 0 to 0.001 (MAE).

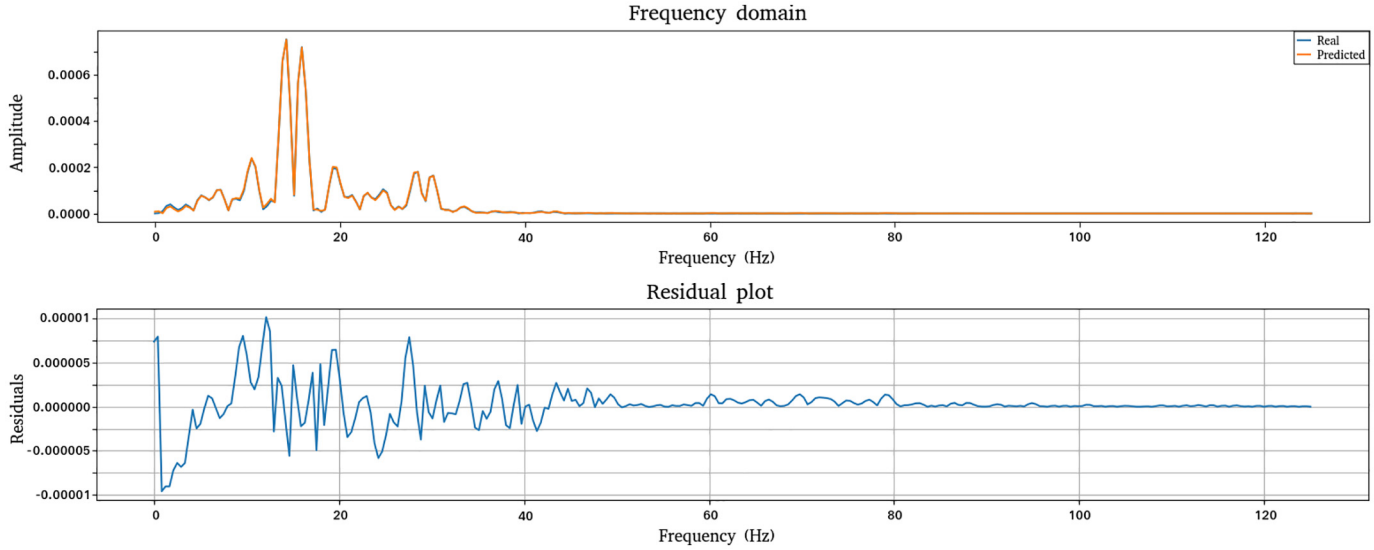

Fig. 8. Reference dataset (in blue), predicted data (in orange) and residual of 1000 CSG panels in frequency domain.

was not able to predict reflections because they did not affect the loss function enough.

In order to sort this problem out we redefined the loss function as

$$loss = \|y_{true} - y_{pred}\|_1 \cdot G \tag{1}$$

with G defined as a quadratic function $G = \frac{x^2}{\tilde{x}}$ where x is the two-way traveltime $\tilde{x}$ the median on the whole domain of the function $y = x^2$.

After computing the loss function, the error is backpropagated and the weights on each neuron are updated by using the backpropagation algorithm. For this task we tested several algorithms, namely Adam Kingma and Ba (2014), Adamax Kingma and Ba (2014), AdaGrad Kingma and Ba (2014) and SGD Nemirovski et al. (2009) and we eventually chose a special case of the Adam algorithm, AdaMax Kingma and Ba (2014), which is the most used for regression tasks and exhibits better performance for this problem. We tested also different batch sizes and we got best stability in loss function with batch size 512.

2.3. NN training

We implemented the NN by leveraging on the Keras library Chollet et al. (2015). We trained the model on two NVIDIA Tesla P100 GPU accelerators.

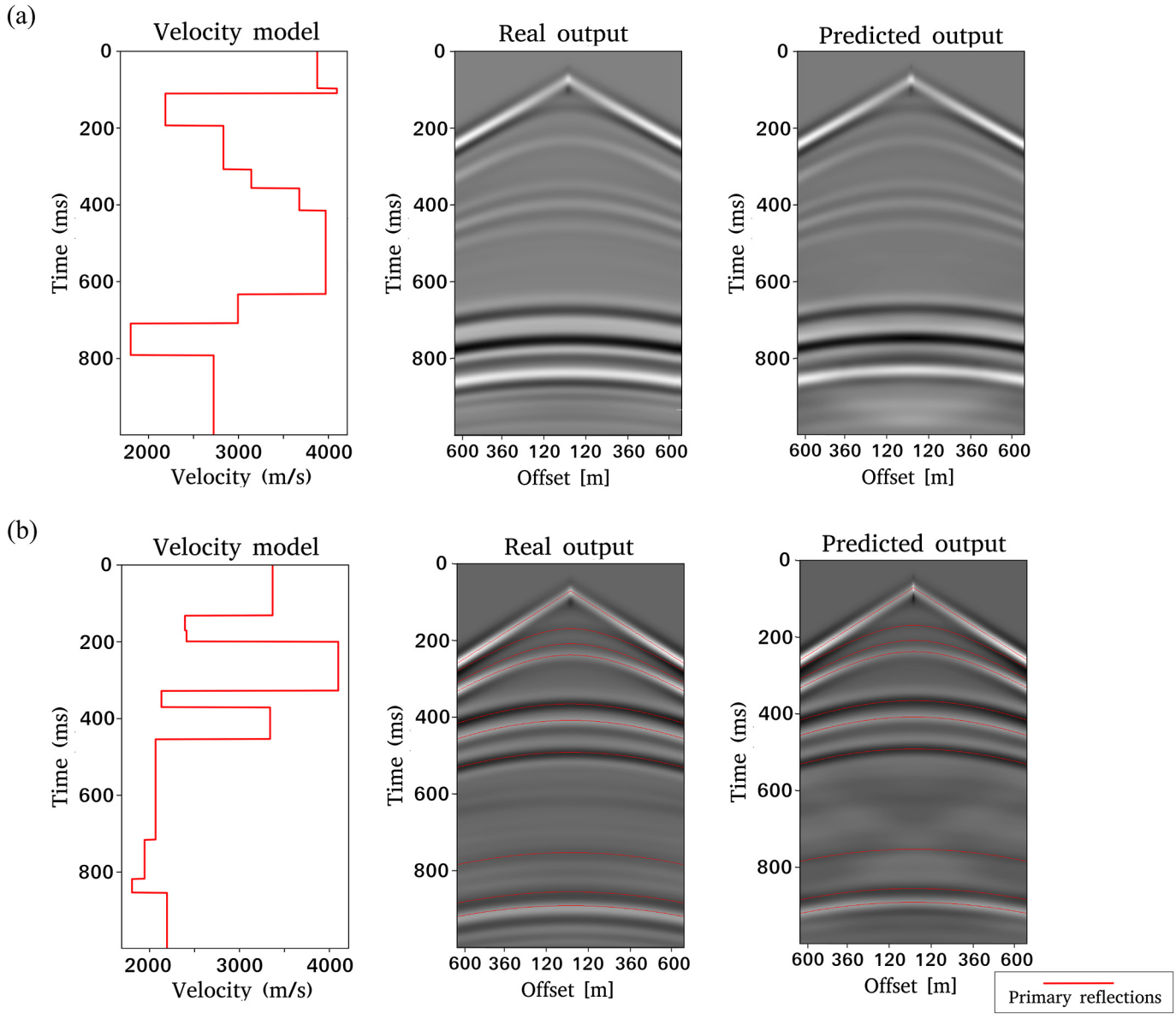

Fig. 9. Two predictions with all 257 offset on two independent profiles. Both are with the input profile, on the left, the desired output, in the middle, and the predicted output, on the right.

All the images shown in this paper are predicted from blind data, which means that the net was not trained on them.

Hidden layer geometry is the same, but we used different numbers of neurons in the output layer to reproduce the number of geophones.

Fig. 3 shows the structure of the training process with the NN seen as a black box. We split the generated dataset, made of 10,000 samples, in training and validation datasets (80% the former and 20% the latter). With samples we are referring to a the vector of velocity values and its associated CSG panel. The proposed model is based on few weights respect to Mosley's model, so we decided to use half dataset. Data selection and splitting is done randomly and no specific samples were selected.

A major drawback of such approach is the limited flexibility, i.e. the trained NN is able to predict a number of different situations but source-receiver offset spacing and input wavelet are limited to the ones it was trained with. This can overly extend the training time and reduce the profitability of the method.

Therefore, we developed an alternative strategy by retraining the model, first trained with one of the two test datasets, with smaller datasets that included new features to avoid the need of a complete training for each modified feature.

Due to the fact that the NN has to internally encode information like input wavelet and offset spacing into its weights, a change of simulation parameters should not vary deeply the main distribution of weights.

By doing so the function to be minimized by the NN should be more linear Kavzoglu (2009) and it should need less epochs to converge to a good result.

The training process on MOS dataset is shown in Fig. 4, in which the loss is plotted against epochs of training, we applied Early stop criteria as Keras Callback. The training process on DEV dataset is shown in Fig. 5: validation and training loss do not diverge too much and they are close enough to each other, which means prediction is good and it is not overfitting.

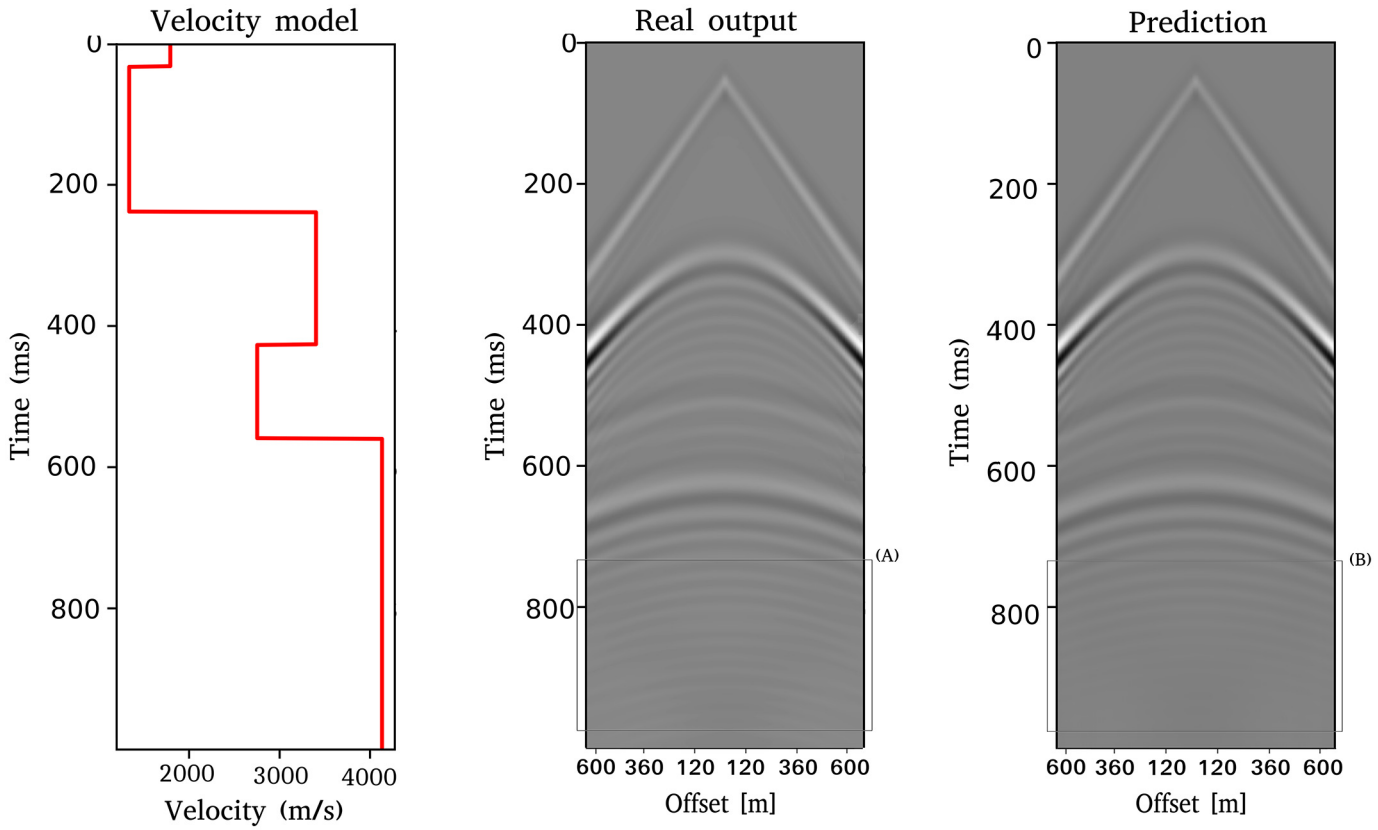

Fig. 10. Reference output and prediction of a CSG panel with numerical dispersion problems due to Finite Differences grid dispersion (A and B).

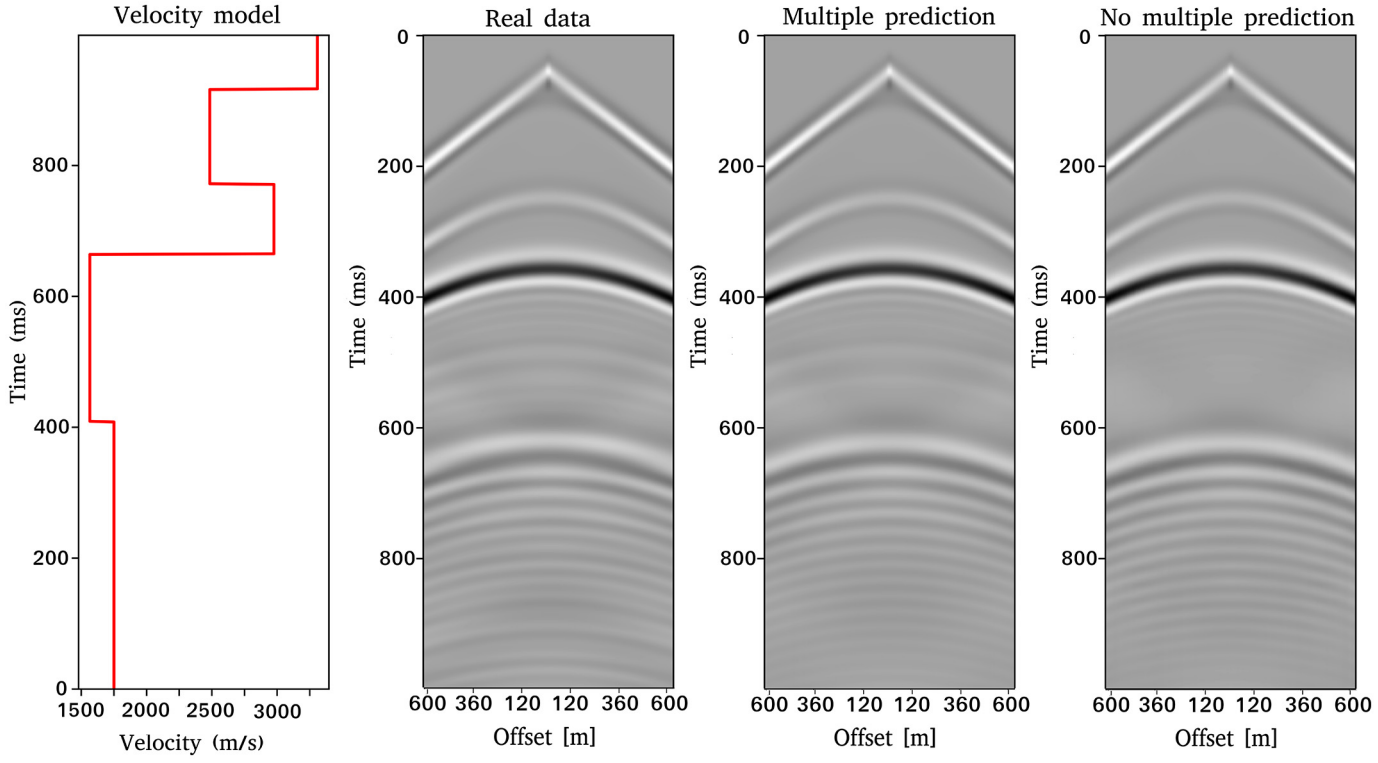

Fig. 11. Input, reference output, prediction of the retrained model and prediction of the original model.

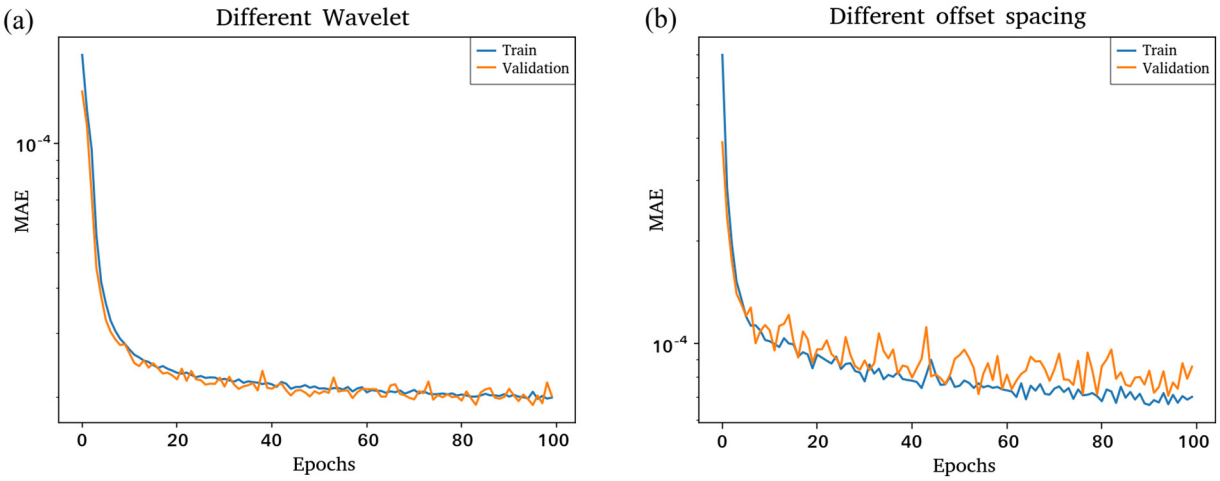

Fig. 12. Loss vs epochs for the retraining of a model with a different waveform (a) and different offset spacing (b).

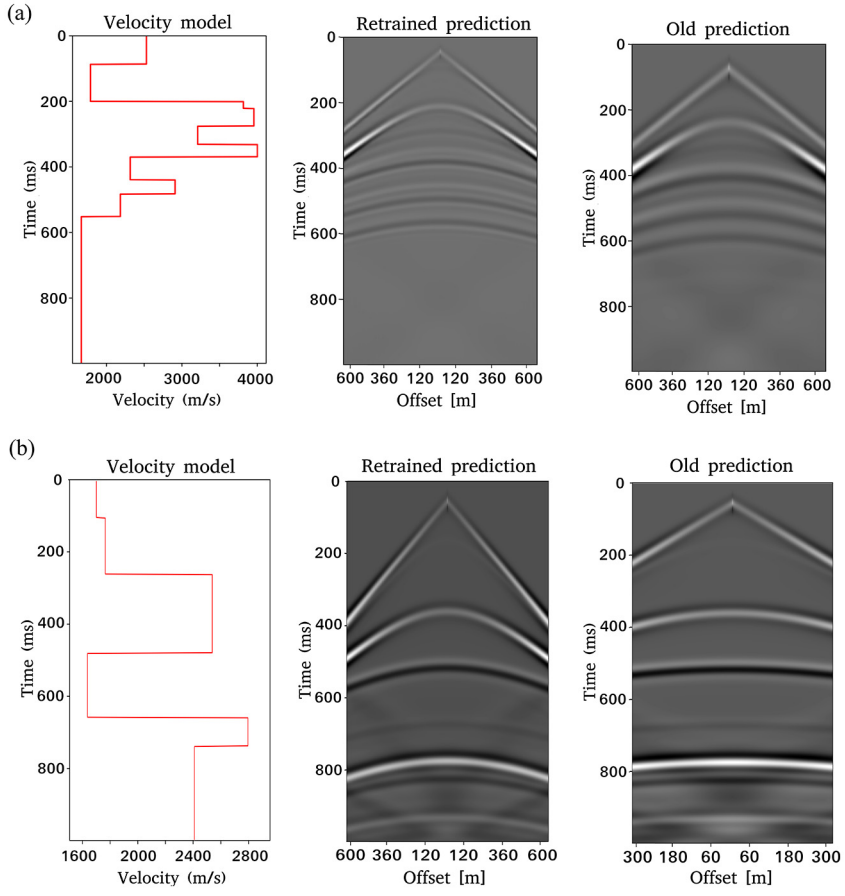

Fig. 13. Prediction of an independent velocity function of the retraining of a model with a different waveform (a) and different offset spacing (b). Old prediction (on the right) and the new prediction (in the middle).

3. Results

3.1. MOS dataset

After the training phase, we tested the net on the dataset generated by Moseley et al. Moseley et al. Moseley et al. (2019a) to evaluate the capability of the net to generalize the problem, i.e. to make correct predictions from datasets different from the training ones.

In this case the training lasted 40 h on two NVIDIA Tesla P100 GPU accelerators and the loss decreased quite smoothly.

After the training, we tested the NN on a case not utilized in the training phase: an example of this prediction and of the corresponding

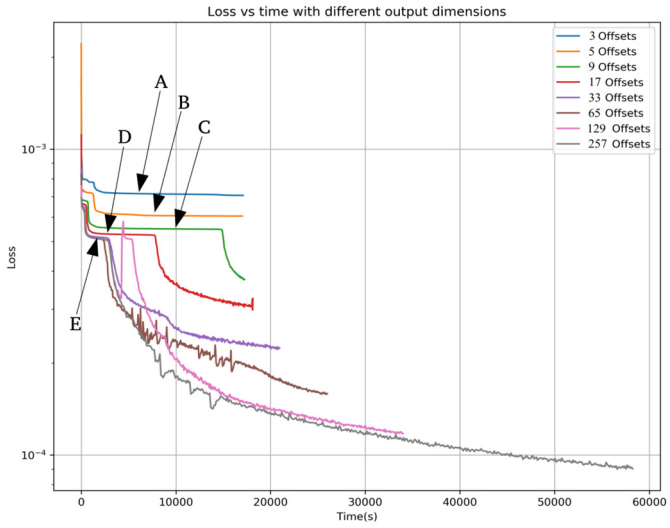

Fig. 14. Loss vs time for different offsets. We can notice a plateau at $(5-8)\times 10^{-4}$. With letter A, B, C and D we referred to the plateau of, respectively, 3,5, 9, 17 offsets. With letter E we show the plateau of the remainder data.

residuals is shown in Fig. 6. The direct wave is well predicted; there are still small errors in the magnitude of the gained reflections, but the peak positions are correctly predicted.

In Fig. 6 we can split the analysis of the results in two parts, focusing the part from 0 *ms* to 350 *ms* and from 350 *ms* to the end.

The first part of the data contains primary reflections and we find some small phase shifts, in red boxes in Fig. 6, but the prediction is good. It is able also to accurately reproduce interference between signals.

The second part is characterized by the presence of multiple reflections only and noise and no primary reflections: in this case the NN does not reproduce anything. This problem can be found in all the dataset and it is present also in the Moseley et al. model Moseley et al. (2019a). This lack of prediction is due to an inadequate statistics on such event in the training dataset.

In order to evaluate prediction performances of NN, we predicted 1000 panels from the dataset and we analyzed the error by plotting the residuals of both nets on a histogram with a logarithmic vertical axis (Fig. 7) and the horizontal axis divided in 50 bins linearly spaced from 0 to 0.001: we can see that using LSTM we got a lower error. We further compared the reference data and the results of prediction of our net in the frequency domain. In Fig. 8 we can see the amplitude spectrum of reference data (in blue) and predicted data (in orange) and residual of 1000 CSG panels. Results are satisfactory as the mean absolute error is 0.89%.

3.2. DEV dataset

We trained the net on another dataset obtained from numerical simulation to have better control of the training process. That allow us to study in depth the importance of the dataset and test different critical situations. In order to do this, we kept the offset spacing and wavelet fixed, but we varied the number of layers in the 1-D model.

In Fig. 9 are presented two predictions with 256 offsets on two independent velocity profiles: we show the input profile on the left, the desired output in the middle, and the predicted output on the right.

In Fig. 9-a, prediction is very close to the target, except for the central area at 950 *ms*, in which we find a small positive amplitude error.

In Fig. 9-b, we have a CSG with multiple reflections: we have plotted the 2-way traveltime of the primary reflections over the predicted data, so that we can easily identify non-primary reflections. We can see a multiple reflection predicted from the model at 600 *ms*.

In the panel we can also see a signal, similar to the one generated by reflection from edges, reproduced by the net. Prediction time for 257 offset CSG (DEV dataset) is 0.064 s.

3.3. The role of the training dataset

Because the NN does not model data from equations, it has to learn from the output of a forward model how an input of the forward model is related to its response. This leads to a crucial role of the dataset used: a bad dataset will lead to a bad model as the net will learn all the features that are in training data.

In Fig. 10, we have a critical level of numerical dispersion, marked with A and B in Fig. 10. Despite this, the net is able to accurately reproduce such phenomena.

It is interesting that such predictions come from the same trained net that made the prediction shown in Fig. 9: this means that the net is also able to discern when and how dispersion has to be predicted. The NN can discern different scenarios and predicts numerical dispersion quite accurately. The model had problems in predicting multiple reflections, as presented in the previous section, and we found a problem due to inadequate statistics on such events in the training dataset.

At the beginning the model was trained on a subsurface model with 7 layers: this lead to a very low probability in finding multiple reflections and the net seemed not to learn what these signals were due to. In order to make an event important in a dataset we had to find a lot of examples of this, and it has to be of significant amplitude compared to others: in this dataset multiple reflections were too sparse and of low amplitude to affect it significantly.

To try to solve this problem, we retrained the already trained model on a dataset generated on purpose to contain several multiple reflections. We imposed a 3-layers subsurface model with the first two layers closer to each other than the third one. This led to a dataset with lots of statistics on multiple reflections.

In Fig. 11 we can see: the input, the reference output, the prediction of the retrained model and the prediction on the original model.

As we can see in the retrained model prediction, a multiple reflection at 500 *ms* is close to the reference signal, while the old net did not predict this signal.

3.4. Retraining

In order to improve the applicability of the proposed NN, we had to make it more flexible by allowing rapid inclusion of information like waveform and offset spacing.

A short scheme of this process follows:

1. Generate a new small dataset ~10% of the dimensions of the first dataset with the new features.
2. Train the previously trained model on the new dataset.

As we can see from loss vs epochs plot, in Fig. 12, the new training converges quickly to a minimum and the space in which the gradient is computed seems to be smooth.

It is apparently not necessary to create new set of weights, because the old prediction is already in the range of the new right solution: a limited amount of additional work is enough to reach the best solution as it reaches a minimum in less than an hour.

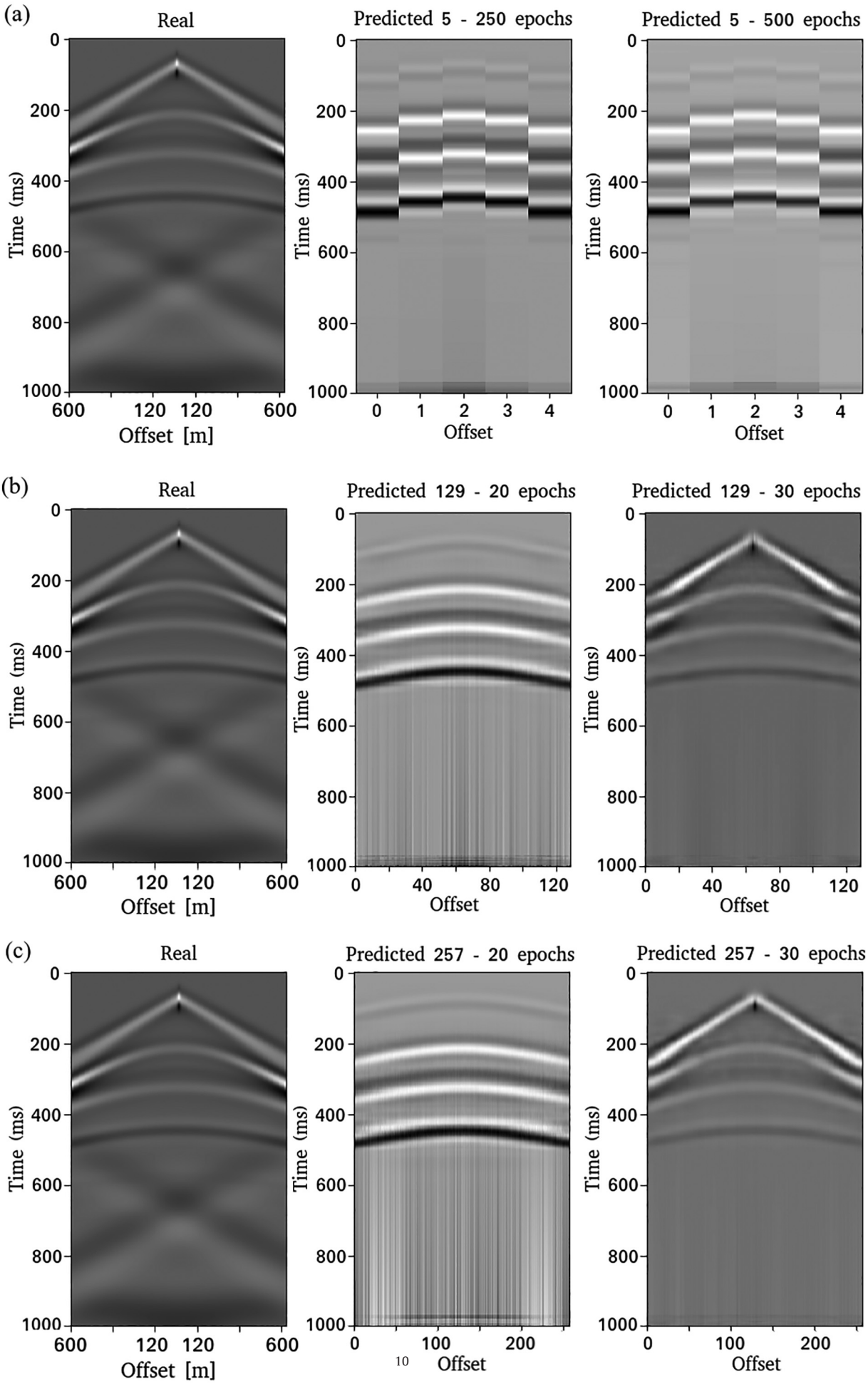
(a)
Real
Predicted 5 - 250 epochs
Predicted 5 - 500 epochs
Time (ms)
Offset [m]
Offset
(b)
Real
Predicted 129 - 20 epochs
Predicted 129 - 30 epochs
Time (ms)
Offset [m]
Offset
(c)
Real
Predicted 257 - 20 epochs
Predicted 257 - 30 epochs
Time (ms)
Offset [m]
Offset

Table 2
Total training time, the moment where the problem starts to be solved and prediction time for each number of offsets considered.

Offsets	Total time (h)	Solving time (h)	Prediction time (s)
3	4.80 h	not solved	0.017 s
5	4.74 h	not solved	0.015 s
9	4.80 h	4.00 h	0.015 s
17	5.05 h	2.16 h	0.019 s
33	5.83 h	0.80 h	0.021 s
65	7.24 h	0.60 h	0.024 s
129	9.46 h	0.80 h	0.033 s
257	16.21 h	0.80 h	0.064 s

Fig. 13-a compares the old prediction (on the right) and the new prediction (in the middle), after the retraining, with a different waveform. While in the old model the source wavelet was a Ricker with peak frequency 10 Hz, in the retrained model we used a higher 20 Hz frequency.

As we have done for the waveform, we have trained a model also to predict a different offset spacing: the results are shown in Fig. 13-b.

3.5. Computational complexity

We measured the time required to train 500 epochs as a function of the dimension of the desired output. To do this we trained the same net increasing the number of columns (offsets) of the output matrix as $2^n + 1$.

In order to evaluate the net performance, we checked also the trends of the loss as a function of time: this is plotted in Fig. 14.

As we can see we have different trends but all of them show a plateau. If we analyze the behavior of the function in the surroundings of such plateau we find out that it is the point at which the net really learns to solve the problem, as shown in Fig. 15.

Up to a 9-offset threshold the net is not able to solve the problem in 500 epochs; then it seems to learn faster when the number of offsets increases.

Total training time and time required to solve the problem are reported in Table 2.

In Fig. 15 we can see the exact timing when the problem was understood with 5, 129 and 257 offsets. We got this timing by looking at the predictions made during the training of the NN at the time the loss exited the plateau. We can also see that 500 epochs are not enough to solve the problem if the number of offset is 5.

In order to evaluate prediction time we used each trained model to predict 100 panels and we took the average time for each model. Results are shown in Fig. 16. Prediction time for a 257 offset CSG panel, under constraints shown in Table 1 and with the same hardware, is of 0.08 s compared to 2.17 s of the FD algorithms used in this paper. According to this benchmark, this would lead to a method that is 27 times faster than the classical one.

Even though prediction time is smaller compared to that obtained by classical methods, it is affected by different factors.

Prediction time for this net depends only on the number of offsets used, on temporal discretization and on record length in time.

The FD generation time, instead, does not depend on number of offsets but on model extension and discretization.

4. Conclusions

The potentialities of Neural Network (NN) in the solution of non linear problems are well known and the successful application to the present case, i.e. synthetic seismic data generation in the 1-D acoustic case, demonstrates such potential.

The proposed methodology is robust and can solve the forward problem, i.e. numerical modelling of seismic wave propagation starting from the velocity model faster than classical methods, especially when high-frequency source wavelets are considered.

For the 1-D acoustic case NN are a good alternative to the classical Finite Difference (FD) method. They are able to rapidly and accurately solve the problem.

The waveform is well reproduced and the net manages to predict multiple reflections.

While prediction time is much lower than classical methods (e.g. FD), the training process needs large times and datasets. Such large training effort may imply that the applicability of the method is limited.

In order to tackle such issue and increase flexibility and applicability we propose a strategy based on retraining.

The proposed method gives good results and is able to make adjustments of the NN parameters in a short time (less than one hour on two NVIDIA Tesla P100 GPU accelerators) and with a small dataset.

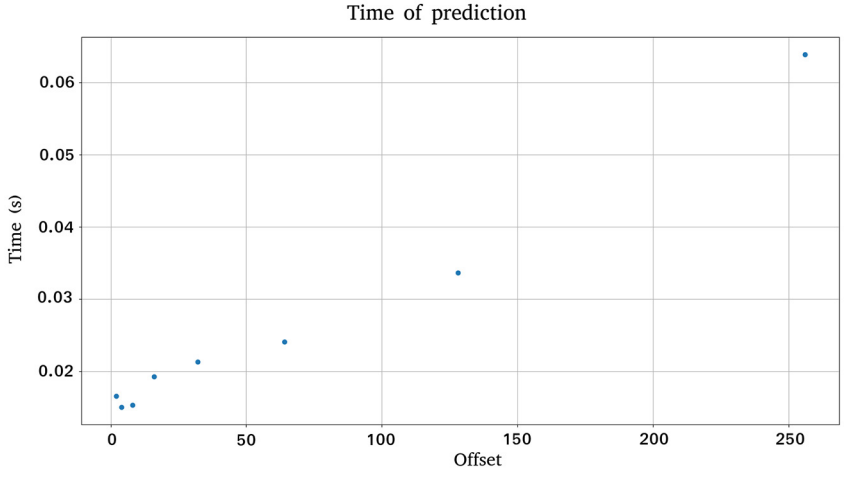

Fig. 16. Prediction time vs offsets. Data are reported in Table 2.

Fig. 15. Comparison between synthetic (FD simulation, column left) and predicted data for 5 (a), 129 (b) and 257 (c) offsets.

In this way, even though we are still linked to FD data generation, we are able to modify parameters in the NN by using just a small amount of data generated by FD. Our results show that the proposed method allows fast and easy parameter change.

The specific architecture and the use of LSTM allowed to obtain seismograms with lots of offsets without substantially increasing the training time.

One possible future work is the application of this methodology for the inverse problem solution and may offer great benefits in the solution of challenging cases, such as e.g. velocity function inversion in presence of multiple reflections.

This technique may open new perspectives of development, in particular the extension to the 2-D and 3-D elastic case. Due to the performance demonstrated in the present work, such extension may be feasible and they can lead to efficient alternative to classical methods.

Nonetheless, the complexity of such extension will require a complete revision of the design of the net.

Author statement

G. Roncoroni: Conceptualization, Methodology, Software, Writing - original draft.

N. Bienati: Conceptualization, Supervision, Writing - review & editing

C. Fortini: Conceptualization, Supervision, Software, Writing - review & editing.

L. Bortolussi: Conceptualization, Supervision, Software, Writing - review & editing.

M. Pipan: Conceptualization, Supervision, Writing - original draft, Writing - review & editing, Project administration.

Declaration of Competing Interest

None.

Acknowledgments

The authors would like to thank Eni, for the opportunity to pursue this work, also through the use of the computational resources made available at ENI's HPC4,[1] and for the approval to bring it to publication. The authors also thank the Landmark's University Grants Program (UGP) and the National Research Program in Antarctica of the Italian Ministry of University and Research for the additional support

References

Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., Shelhamer, E., 2014. Cudnn: Efficient primitives for deep learning. CoRR abs/1410.0759. URL. http://arxiv.org/abs/1410.0759.

Chollet, F., et al., 2015. Keras. https://keras.io.

Glorot, X., Bordes, A., Bengio, Y., 2011. Deep sparse rectifier neural networks. In: Gordon, G., Dunson, D. (Eds.), Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. PMLR, pp. 315–323.

Guo, R., Zhang, J., Liu, D., Zhang, Y., Zhang, D., 2019. Application of Bi-Directional Long Short-Term Memory Recurrent Neural Network for Seismic Impedance Inversion. pp. 1–5 URL.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9, 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

Kavzoglu, T., 2009. Increasing the accuracy of neural network classification using refined training data. Environ. Model Softw. 24, 850–858.

Kingma, D., Ba, J., 2014. Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.

Louboutin, M., Lange, M., Luporini, F., Kukreja, N., Witte, P.A., Herrmann, F.J., Velesko, P., Gorman, G.J., 2019. Devito (v3.1.0): an embedded domain-specific language for finite differences and geophysical exploration. Geoscientific Model Development 12, 1165–1187 URL:. https://www.geosci-model-dev.net/12/1165/2019/ https://doi.org/10.5194/gmd-12-1165-2019.

Moseley, B., Markham, A., Nissen-Meyer, T., 2019a. Fast Approximate Simulation of Seismic Waves with Deep Learning.

Moseley, B., Markham, A., Nissen-Meyer, T., 2019b. Seismic Simulation Wavenet. URL. https://github.com/benmoseley/seismic-simulation-wavenet.

Nemirovski, A., Juditsky, A., Lan, G., Shapiro, A., 2009. Robust stochastic approximation approach to stochastic programming. Society for Industrial and Applied Mathematics 19, 1574–1609.

Oord, A.V.D., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K., 2016. Wavenet: A generative model for raw audio. URL. http://arxiv.org/abs/1609.03499.

Øye, O., Dahl, E., 2019. Velocity model building from raw shot gathers using machine learning. https://doi.org/10.3997/2214-4609.201900039.

Sherstinsky, A., 2020. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. Physica D: Nonlinear Phenomena (0167-2789), 132306 https://doi.org/10.1016/j.physd.2019.132306. https://www.sciencedirect.com/science/article/pii/S0167278919305974.

Simard, P.Y., Steinkraus, D., Platt, J.C., 2003. Best practices for convolutional neural networks applied to visual document analysis. Seventh International Conference on Document Analysis and Recognition. Proceedings, pp. 958–963 https://doi.org/10.1109/ICDAR.2003.1227801.

Yang, F., Ma, J., 2019. Deep-learning inversion: A next-generation seismic velocity model building method. GEOPHYSICS 84, R583–R599. https://doi.org/10.1190/geo2018-0249.1.

[1] HPC4 is a supercomputer owned by ENI spa that has a peak performance of 18.6 Petaflops