Stacking velocity estimation using recurrent neural network

Reetam Biswas, University of Texas at Austin; Anthony Vassiliou, Rodney Stromberg, GeoEnergy Inc. and Mrinal K. Sen, University of Texas at Austin*

SUMMARY

We describe a new method based on the Machine Learning (ML) technique for normal moveout correction (NMO) and estimation of stacking velocity. A Recurrent Neural Network (RNN) is used to calculate stacking velocity directly from the seismic data. Finally, this velocity is used for NMO correction of the data. We used the Adam optimization algorithm to train the network of neurons to estimate stacking velocity for a batch of seismic gathers. This velocity is then compared with the correct stacking velocity to update the weight. The training method minimizes a cost function defined as the mean squared error between the estimated and the correct velocities. The trained network is then used to estimate stacking velocity for rest of the gathers. Here we illustrate out method on a noisy real data set from Poland. We first trained the network using only 18 percent of gathers and then used the network to calculate stacking velocity for the remaining gathers. We used these stacking velocity to perform Normal moveout correction and finally we stacked to get the post-stack seismic section. We also show comparison between the stacks generated from the two velocities.

INTRODUCTION

Deep neural networks have been applied successfully in several areas of science and engineering, such as handwriting recognition, speech recognition, and signal detection (e.g., Freeman and Skapura, 1991; Cichocki and Unbehauen, 1993). It has recently gained an immense popularity in the field of seismic exploration geophysics. There has been a wide variety of applications of Deep Neural Network. An example of this is first-break picking from seismic data (Murat and Rudman, 1992; McCormack and Rock, 1993) or velocity picking from velocity scans for velocity analysis (Schmidt et al., 1992; Fish and Kusuma, 1994). It has also been applied to the study of shear wave splitting (Dai and MacBeth, 1994) and heavily used in characterization of reservoir from seismic reflection data (An et al., 1993). Some of the recent applications include in classification problems to differentiate one seismic attributes from another (Leiphart and Hart, 2001; Russell, 2004).

We use one of the Machine Learning tools to solve a regression problem where we try to estimate stacking velocity directly from seismic data for Normal Moveout correction. Here both the spatial and temporal information are important for estimation of the stacking velocity. This problem can be solved using a Recurrent Neural Network (RNN). A RNN has the ability to send feedback signals so as to form a directed cycle similar to a Hopfield net (Hopfield, 1982) and long-short term memory (LSTM) (Hochreiter and Schmidhuber, 1997). The network architecture is similar to that of the multilayer feedforward neural network (FNN). The FNN is most widely used to solve pattern recognition problems because of the simplicity and its generality for solving a variety of problems. But a feedforward neural network allows the signal to travel only in one direction, i.e. from input to output. The output of any layer does not affect that same layer, they simply associate input to the output. On the other hand in a recurrent network, the signal can travel in both directions by introducing a loop in the network. These make the network powerful, but it can get extremely complicated. The output of the current state is dependent on the output of the previous states, which gives them a kind of memory. The feedback makes the recurrent network dynamic and their states change continuously until an equilibrium point is reached. If the input is changed then again a new equilibrium needs to be found. The weights are updated using a back-propagation training method, where an update to the weights is calculated by the mean squared error between computed and desired output. The input-output pattern is used for training the network and is termed as training set. The main aim is to learn a mapping from the known input-output patterns and then later apply the trained network on the input with unknown output.

Similar attempt have been made by Calderón-Macı´as et al. (1998) to estimate NMO velocity using a feedforward neural network. In this paper the proposed approach to network training is not restricted to the NMO-correction and velocity estimation process, but it can be adapted to other problems such as prestack migration velocity analysis. We illustrate the method with a real land data set from Poland.

THEORY

This section introduces the various basic concepts about the Recurrent Neural Network (RNN) and describes the way data is input in the network to calculate the stacking velocity.

Recurrent Neural Network (RNN)

A RNN is a type of Neural Network which is quite similar to a feedforward neural network, except that can send a feedback signal, i.e. it also has connections pointing backward. Figure 1 shows an example of a simple RNN. The figure in the left shows the recursive model of the RNN and the right figure shows the unrolled RNN through time. In RNN at time step t every neuron receives both the input vector $\mathbf{x}_t$ and the output vector from the previous time step $\mathbf{y}_{t-1}$ as shown in Figure 1(b). More specifically, suppose a given observation sequence $\mathbf{x} = \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t$ ia associated with the output sequence vector $\mathbf{y} = \mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_t$; we want to examine a map $f : \mathbf{x} \rightarrow \mathbf{y}$. Each neuron in the RNN has two sets of weights: one for the input signal $\mathbf{x}$ represented as $\mathbf{W}_{xy}$ and another for the output from the previous time step $\mathbf{y}$ represented as $\mathbf{W}_{yy}$. We can represent the network using

$$\mathbf{y}_t = \phi\left(\mathbf{W}_{xy}^T \cdot \mathbf{x}_t + \mathbf{W}_{yy}^T \cdot \mathbf{y}_{t-1} + \mathbf{b}\right), \tag{1}$$

where ϕ represents the activation function and $\mathbf{b}$ is the bias vector. There are several choice of activation available e.g., sigmoid, RELU, tanh etc. Generally, in our application we use RELU activation function. Notice that the one hidden layer of RNN can be easily extended into multiple layers, which represents a Deep Neural Network. Having multiple neurons helps to characterize multiple features in the data and thus predict the output better.

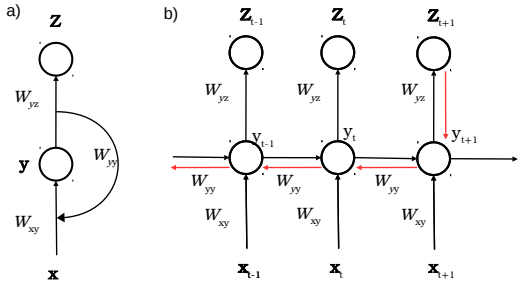

Figure 1: Shows a simple example of a RNN a) the left diagram shows the recursive form of RNN, and b) the right shows the extended form of RNN in time.

During training of the network, in a single iteration we update the weight for not just a single sequence but a multiple sequence of data known as mini-batch. Equation 1 can be modified to compute the output of the whole mini-batch in a single shot by representing it as

$$\begin{aligned} \mathbf{Y}_t &= \phi\left(\mathbf{X}_t \cdot \mathbf{W}_{xy} + \mathbf{Y}_t \cdot \mathbf{W}_{yy} + \mathbf{b}\right. \\ &= \phi\left(\begin{bmatrix}\mathbf{X}_t & \mathbf{Y}_{t-1}\end{bmatrix} \cdot \mathbf{W} + \mathbf{b}\right), \end{aligned} \tag{2}$$

with $\mathbf{W} = \begin{bmatrix}\mathbf{W}_{xy} \\ \mathbf{W}_{yy}\end{bmatrix}$. If the mini-batch has m instances of different sequences containing n_n neurons and n_i input vector size; then matrix $\mathbf{Y}_t$ has a dimension of $m \times n_n$, $\mathbf{X}_t$ has $m \times n_i$, $\mathbf{W}_{xy}$ has $n_i \times n_n$, $\mathbf{W}_{yy}$ has $n_n \times n_n$ and finally the bias vector $\mathbf{b}$ has dimension of n_n. Note that $\mathbf{Y}_t$ is a function of $\mathbf{X}_t$ and $\mathbf{Y}_{t-1}$, which again is a function of $\mathbf{X}_{t-1}$ and $\mathbf{Y}_{t-2}$ and so on. Thus $\mathbf{Y}_t$ is a function of all the inputs since time $t = 0$. The value of $\mathbf{Y}_0$ is typically set to zero. Due to this dependence of recurrent neuron on the previous output, it exhibits a memory property. A part of the network uses the memory of the previous inputs to predict the future output. This simple cell is also called as a memory cell.

The output vector $\mathbf{y}$ from RNN is a vector of size n_n, but we can modify the length to a desired length by applying a Fully Connected layer on top of the RNN. This can be represented as

$$\mathbf{Z}_t = \texttt{FC}(\mathbf{y}_t), \tag{3}$$

where the fully connected layer FC has a weight $\mathbf{W}_{fc}$ of dimension of $n_n \times k$; where k is the number of desired output. Now, after a single forward pass for a mini-batch we use mean squared error of the predicted velocity and the given velocity as

$$E = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{Z}_{given} - \mathbf{Z}_{predicted})^2. \tag{4}$$

We employ Adam-optimization to update the weights after each forward pass of a mini-batch. The gradient $\frac{\partial E}{\partial w}$ is calculated using back-propagation and the weights are updated using a given learning rate value, which acts as a step length for the update. For detail about the back-propagation technique refer to Chen (2016). Note that here $\mathbf{Z}$ represents the stacking velocity.

Stacking velocity Estimation

In a CMP gather there can be multiple hyperbolas spanning a large offset and in temporal direction, therefore estimating correct stacking velocity such that it places the reflection at the correct location is challenging and important. The main aim is to flatten the hyperbolas so that the reflection from same depth (TWT) can be stacked together constructively. For estimating the velocity at a particular time step, temporal as well as spatial information from the nearby time step and offset is quite important. Therefore to estimate stacking velocity at a particular time step we have used a window, spanning the whole offset range (NX) and $2N$ in the temporal direction. Figure 2 shows the representation, where the dimension of the window (represented in red) used is $2N \times NX$, for estimating stacking velocity at the magenta location.

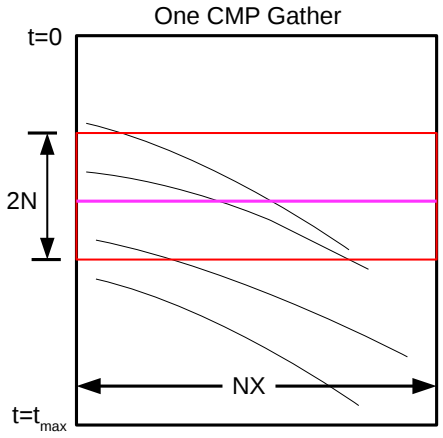

Figure 2: Representation of a CMP gather (with offset size of NX) and the blocks of data from the CMP gather used for creating a single instance of a mini-batch. The magenta line represents the time at which velocity is being estimated and the red block (of size $2N$) represents the data used for estimating the velocity at that point.

RESULT

We applied our RNN based stacking velocity estimation on a Prestack 2D land vibroseis data provided by Geofizyka Torun Sp. Z.o.o, Poland available in public domain. The data after initial pre-processing and removal of noise is ready for velocity analysis. In our test case we have used 1000 neurons in the recurent network, 60 offsets in each gather, window size of 100 and a total of 700 samples in temporal domain. The data has a sampling rate of 2ms. We initially generated stacking velocity using semblance based velocity analysis for a small part of the section. Figure 3a shows the picked stacking velocity from the velocity analysis. This is treated as known or given velocity. We then divided the seismic gathers randomly in 18 % and 82 % respectively for training and testing set. We used the gathers in the training set and their respective velocity to train the recurrent network. At each iteration a mini batch is created using a gather and weights are updated. We ran multiple epoch of 10 to train the network and gain desired error level. After the network is trained, the velocity is estimated on the test data. Figure 3b shows the predicted stacking velocity from the recurrent network. This section has both the velocity from training and testing.

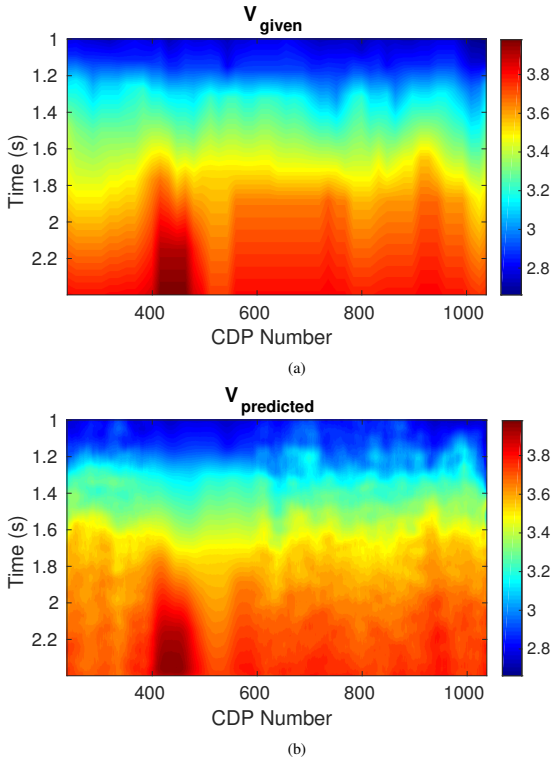

Figure 3: a) shows the given staking velocity and b) shows the estimated stacking velocity from the RNN.

After the velocity has been calculated for the part of the data, we used these velocities for the NMO correction of the data and finally stack the data to check the correctness of the estimation. Figure 5 shows the stacked section, where the first stack is generated using the given velocity and the second stack is generated using the network estimated stacking velocity. Just by observation they seem quite similar. Figure 6 shows an example of velocity estimation from a gather selected from test set. The first panel (a) shows the given uncorrected gather where, using an ellipse we have pointed one of the hyperbola in the noisy data. Panel (b) shows the NMO corrected gather using the given velocity and the corresponding hyperbola from panel (a) flattening up. Similarly in panel (c) the gather is corrected using the predicted velocity and the result is similar to the panel (b). Finally we show a comparison of the velocity in panel (d), where the given/true velocity is shown in blue and the estimated velocity in orange.

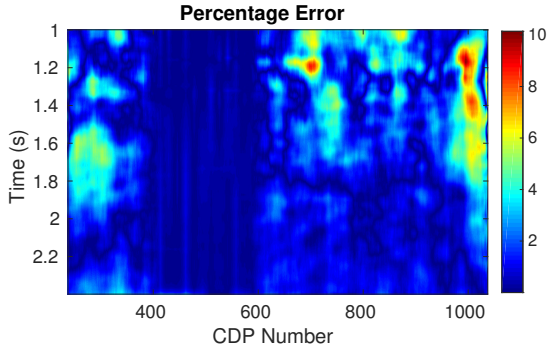

Figure 4: Shows the percentage difference in estimation of $V_{predicted}$ from the the V_{given} NMO velocity

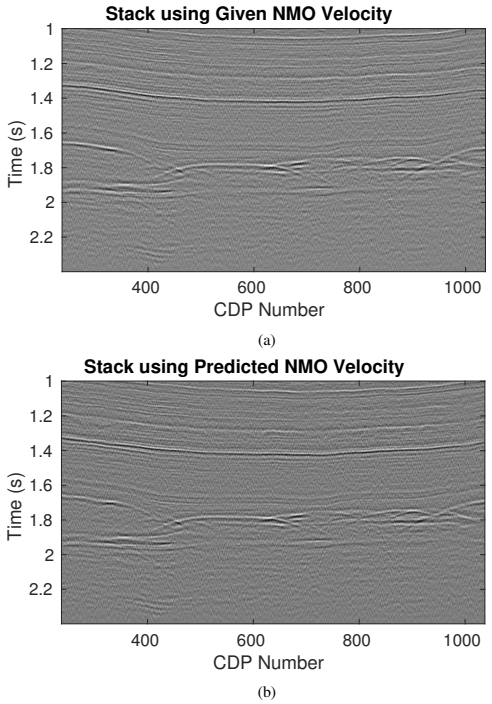

Figure 5: a) shows the stacked section after NMO correction using the given stacking velocity and b) shows the stacked section after NMO correction using the estimated stacking velocity from the RNN.

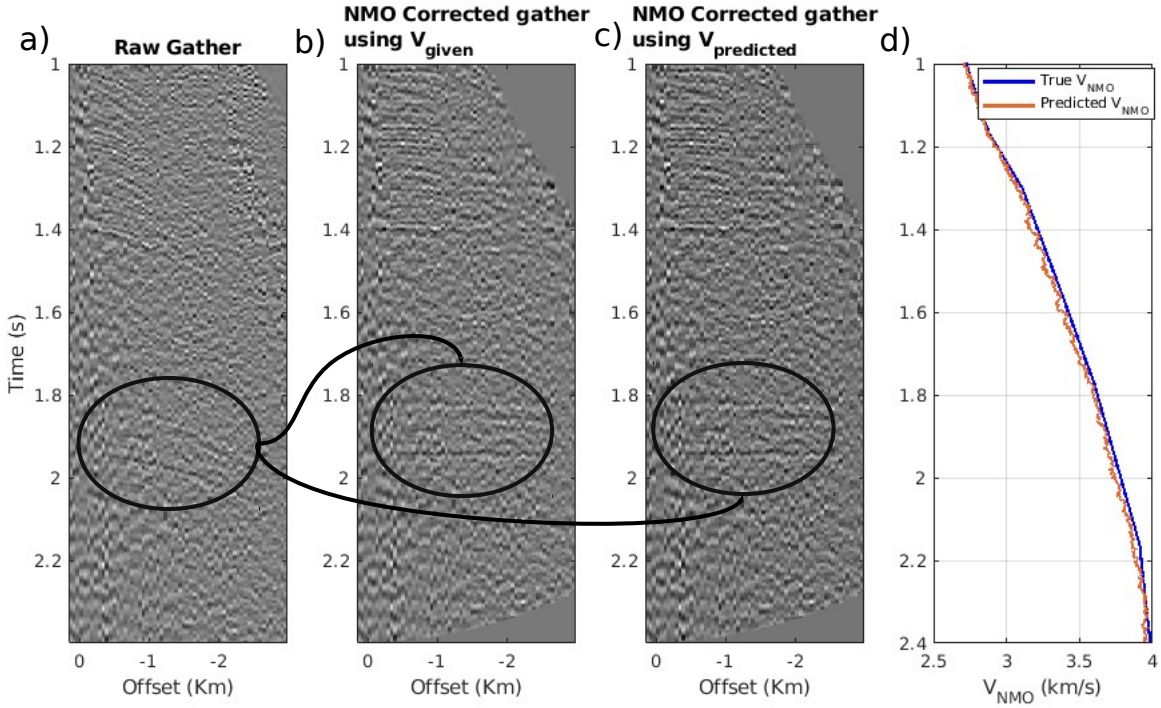

Figure 6: a) Shows a gather before NMO correction at a certain CDP location. One of the hyperbola is highlighted using an ellipse b) Shows NMO corrected gather using the given stacking velocity and the flatten hyperbola is marked by an ellipse. c) Shows NMO corrected gather using the predicted stacking velocity and again the flatten hyperbola is marked by an ellipse. d) Shows the comparison of the given stacking velocity (in blue) and the predicted stacking velocity (in orange).

DISCUSSION AND CONCLUSION

In this paper, we demonstrated one of the applications of machine learning tool in a geophysical problem. We applied Recurrent Neural Network for stacking velocity estimation directly from the seismic gather and performed NMO correction using the estimated velocity. The NMO result is quite similar to the one picked from the semblance based velocity. In stacking velocity estimation, relationship of one point with both the temporal and spatial region is very important. Due to the memory property of the recurrent network, current output depends on the past output, and the neighborhood dependency can easily be established and hence better estimation of stacking velocity. We showed the application of our method on a real data set, where we trained the data using 18% of the data and performed test on the rest of the data after training. The velocity estimate is quite reasonable and match the semblance based velocity.