Inversion and machine learning

Brian Russell

cgg.com

Passion for Geoscience

Introduction

- At the EAGE meeting in London I gave a talk on machine learning applied to inversion at the HPC booth, and was pleased to be invited by Martin to give that talk again in Houston.
- But I have discovered that a big impediment to understanding machine learning is getting inside the “black box” and seeing what is really going on.
- So at the start of this talk I will use a simple geophysical problem to try and demystify machine learning algorithms and show that, like physics-based inversion, they have a mathematical structure that can be understood.
- I will then move to a real-world example and show how to extend our simple single-layer neural network to the deep, or multi-layer, case.
- The results will suggest that we need to use a judicious combination of both the machine learning and physics-based methods.

Theory-based versus machine learning analysis

- A large number of geophysical problems can be linearized and expressed as:

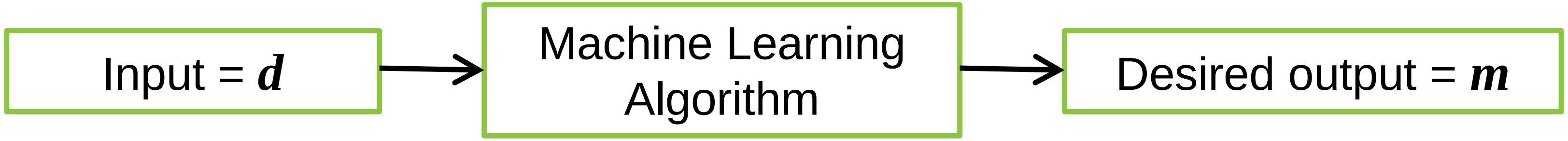
$$\boldsymbol{d} = G\boldsymbol{m}, \text{ where:}$$

$\boldsymbol{d}$ = the input data, $G =$ a geophysical transform, and $\boldsymbol{m}$ = an underlying geological model.

- If we know the parameters in G, the solution can be found using least-squares:

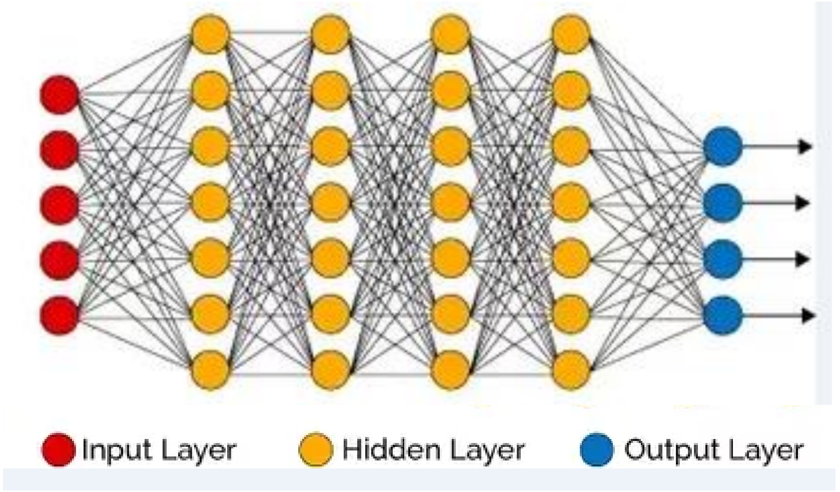
$$\boldsymbol{m} = \left(G^T G\right)^{-1} G^T \boldsymbol{d}$$

- However, in the machine learning approach we present the input and desired output to the algorithm and let it find the relationship:



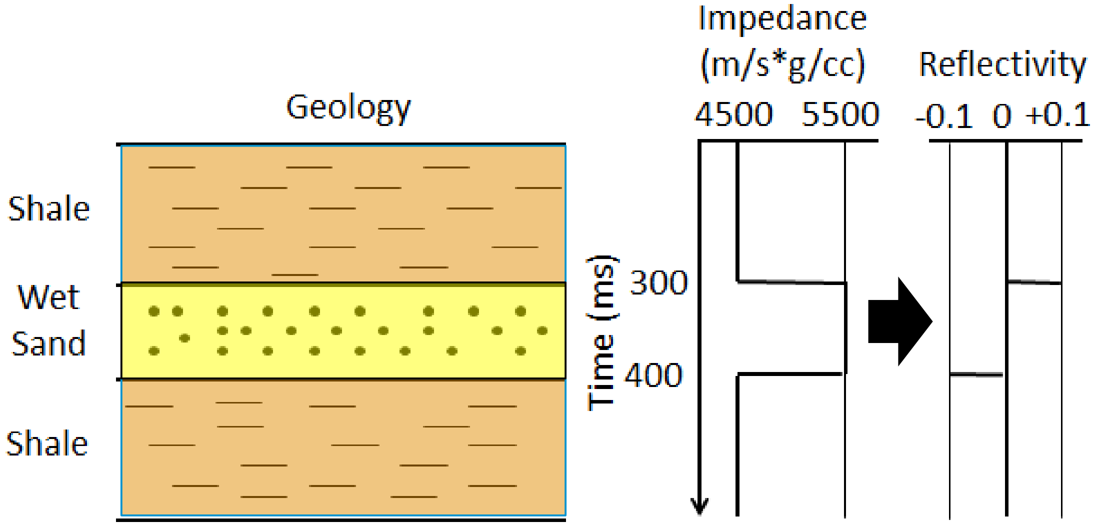
Deep Neural Networks (DNNs)

- The newer machine learning techniques use Deep Neural Networks (DNNs) with many hidden layers.
- In the figure, the inputs are shown as the red circles, the hidden layers as the yellow circles and the outputs as the blue circles.
- The lines between the circles are weights which must be learned by the network.
- But what is a hidden layer, and how do we train the weights?
- I will use a simple model and a network with a single hidden layer to explain this.



The geological model

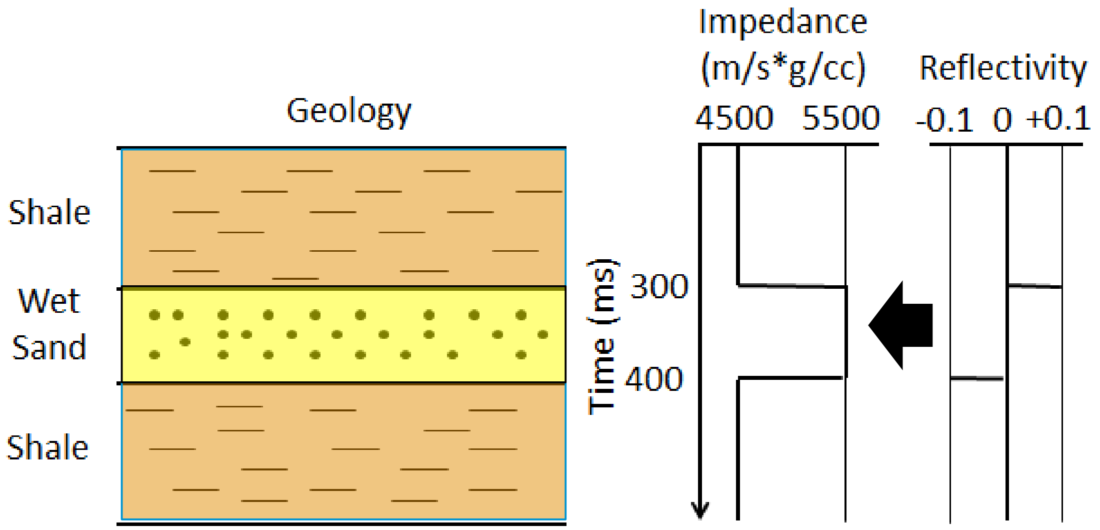
- The model consists of a shale of P-wave impedance $\rho_{sh}V_{sh} = I_{sh} = 4500$ m/s*g/cc, which both overlies and underlies a wet sandstone of impedance $\rho_{ss}V_{ss} = I_{ss} = 5500$ m/s*g/cc, where ρ is density and V is P-velocity.
- The reflection coefficient formula gives two reflection coefficients:



$$r_i = \frac{\rho_{i+1}V_{i+1} - \rho_i V_i}{\rho_{i+1}V_{i+1} + \rho_i V_i} \Rightarrow r_1 = \frac{5500 - 4500}{5500 + 4500} = +0.1 \Rightarrow r_2 = \frac{4500 - 5500}{4500 + 5500} = -0.1$$

Seismic inversion

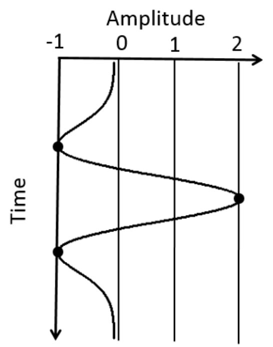
- The two reflection coefficients can be exactly inverted back to impedance using the recursive inversion formula.
- This assumes that the first impedance is correctly estimated.
- The recursive formula is as follows:

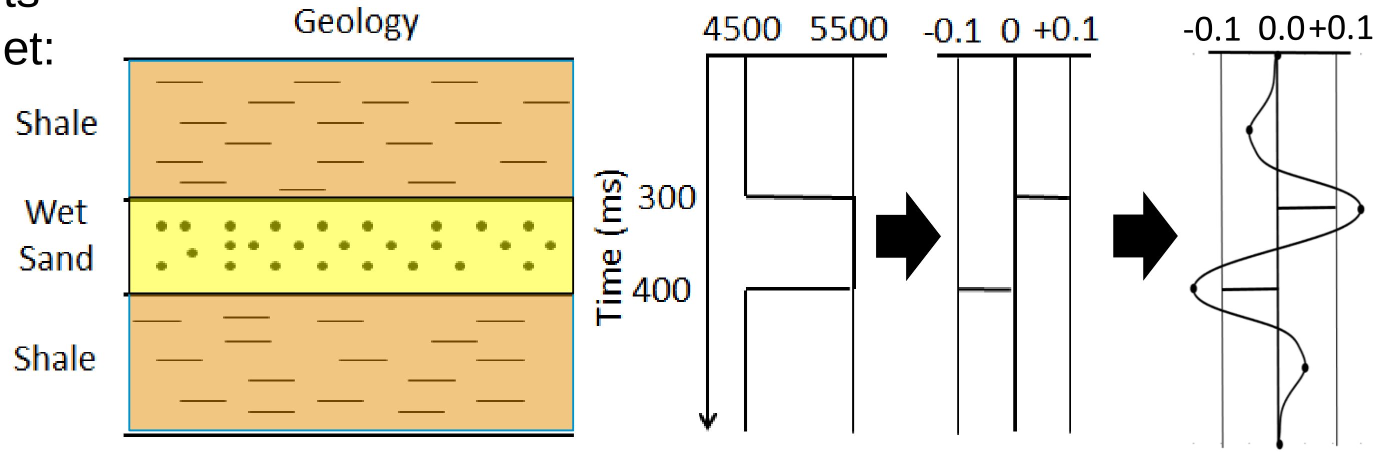


$$r_{i+1} = \rho_i V_i \frac{1+r_i}{1-r_i} \quad \Rightarrow \rho_2 V_2 = 4500\frac{1+0.1}{1-0.1} = 5500 \quad \Rightarrow \rho_3 V_3 = 5500\frac{1-0.1}{1+0.1} = 4500$$

Creating the seismic trace

- We then convolve the reflection coefficients with a Ricker wavelet:





- The convolution equation as follows, where the wavelet has be reduced to 3 points:

$$s = g * r, \text{ where } g = \begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix} = \text{ wavelet}$$

The convolution equation

- The convolution equation can be also be written as a matrix equation:

$$\boldsymbol{s} = G\boldsymbol{r}$$

- In the above equation, $\boldsymbol{s}$ is a vector of seismic values, $\boldsymbol{r}$ is the vector of reflection coefficients given by:

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

- and G is the geophysical matrix with the wavelet in two columns spaced by the number of samples between reflection coefficients:

$$G = \begin{bmatrix} -0.5 & 0 \\ 1 & -0.5 \\ -0.5 & 1 \\ 0 & -0.5 \end{bmatrix}$$

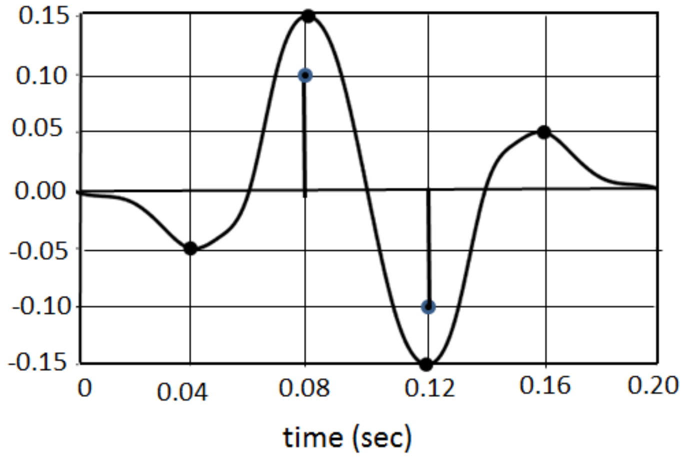
The synthetic trace

- Performing the matrix multiplication, we get the following seismic trace:

$$\mathbf{s} = \mathbf{G}\mathbf{r} = \begin{bmatrix} -0.5 & 0 \\ 1 & -0.5 \\ -0.5 & 1 \\ 0 & -0.5 \end{bmatrix} \begin{bmatrix} +0.1 \\ -0.1 \end{bmatrix} = \begin{bmatrix} -0.05 \\ +0.15 \\ -0.15 \\ +0.05 \end{bmatrix}$$

- This is an example of seismic “tuning” since the wavelets interfere or “tune” to appear as a 90 degree phase wavelet and show an amplitude increase.

- Here is the result of the convolution, where a spline fit has been used to interpolate the values between the spikes:

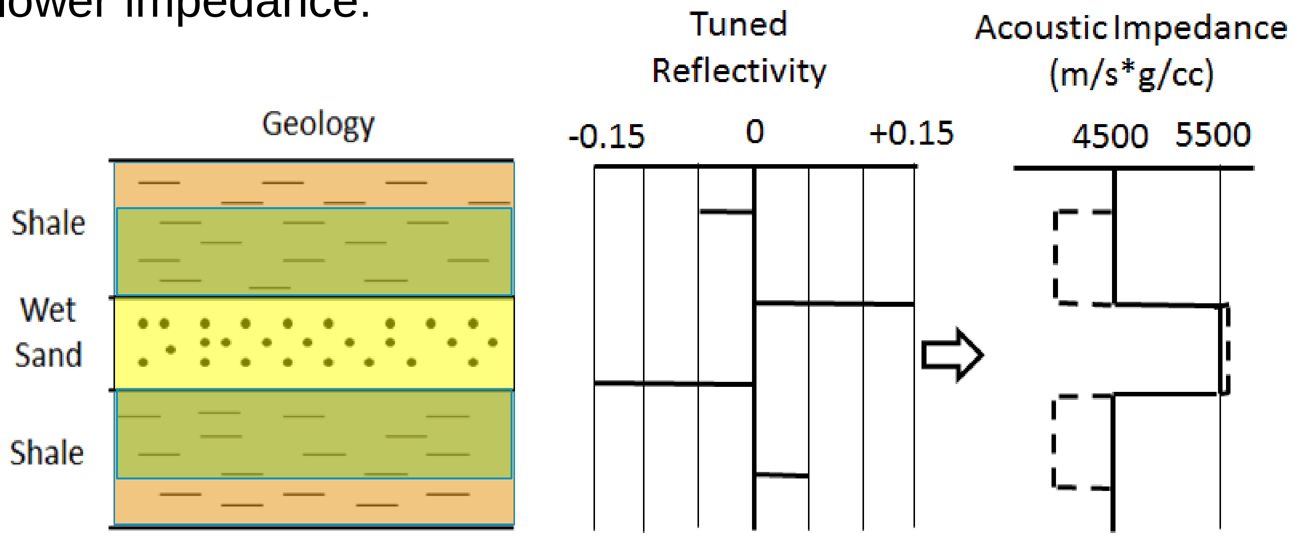


- The original reflection coefficients are displayed, showing the effect of the wavelet.

Inverting the tuned response

- Performing recursive inversion on the tuned seismic response creates two extra layers with lower impedance:

$$\hat{I} = \begin{bmatrix} 4500 \\ 4071 \\ 5508 \\ 4071 \\ 4500 \end{bmatrix}$$



- The “hat” over the impedance indicates that this is only an estimate of the correct answer (the dashed curve is the estimate and solid curve the true impedance).
- Our objective is to extract the true reflectivity from the synthetic seismic trace.

Deconvolution

- The geophysical way to extract the reflectivity is to deconvolve the wavelet.
- Since the G matrix is not square (i.e. there are more knowns than unknowns), this involves the least-squares approach, which is written:

$$\hat{\boldsymbol{r}} = (G^T G)^{-1} G^T \boldsymbol{s} = G^* \boldsymbol{s}, \text{ where } \hat{\boldsymbol{r}} = \text{ the reflectivity estimate,}$$

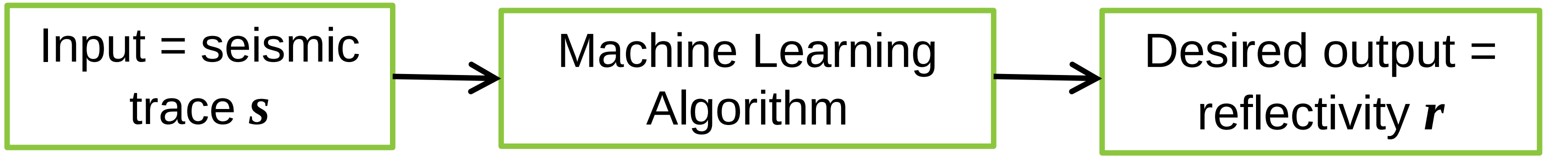
$$\text{and } G^* = \text{the generalized inverse.}$$

- If we know G exactly (which we rarely do!) the answer is perfect:

$$\hat{\boldsymbol{r}} = \boldsymbol{r} = G^* \boldsymbol{s} = \begin{bmatrix} -0.6 & 0.8 & 0.2 & -0.4 \\ -0.4 & 0.2 & 0.8 & -0.6 \end{bmatrix} \begin{bmatrix} -0.05 \\ +0.15 \\ -0.15 \\ +0.05 \end{bmatrix} = \begin{bmatrix} +0.1 \\ -0.1 \end{bmatrix}$$

The machine learning approach

- The deconvolution approach to extracting reflection coefficients from seismic data produces good results as long as we are able to estimate the wavelet.
- Now, let’s look at the machine learning approach, which we will implement as a supervised neural network, where we know both the input and output:


Input = seismic trace ***s***
Machine Learning Algorithm
Desired output = reflectivity ***r***

- That is, we will let the machine learning algorithm learn the weights that will transform the seismic trace into the reflectivity.
- This is actually a type of nonlinear regression, so first we will discuss the linear regression approach.

Linear regression

- In linear regression, we estimate the two unknown weights w_0 (the intercept or bias) and w_1 (the slope) in the equation:

$$\boldsymbol{r} = w_0 + w_1 \boldsymbol{s}$$

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

$$\boldsymbol{r} = S\boldsymbol{w} = \begin{bmatrix} 1 & -0.05 \\ 1 & +0.15 \\ 1 & -0.15 \\ 1 & +0.05 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} 0 \\ +0.1 \\ -0.1 \\ 0 \end{bmatrix}$$

- The column of ones in the S matrix is there to multiply the bias term w_0.

Linear regression

- Since the S matrix is not square, this again involves the least-squares approach, which is written:

$$\boldsymbol{w} = (S^T S)^{-1} S^T \boldsymbol{s} = \boldsymbol{S}^* \boldsymbol{r}, \text{ where } S^* = \text{the generalized inverse.}$$

- Plugging in the values gives: $\boldsymbol{w} = S^* \boldsymbol{r} = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ +0.1 \\ -0.1 \\ 0 \end{bmatrix} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.6 \end{bmatrix}$

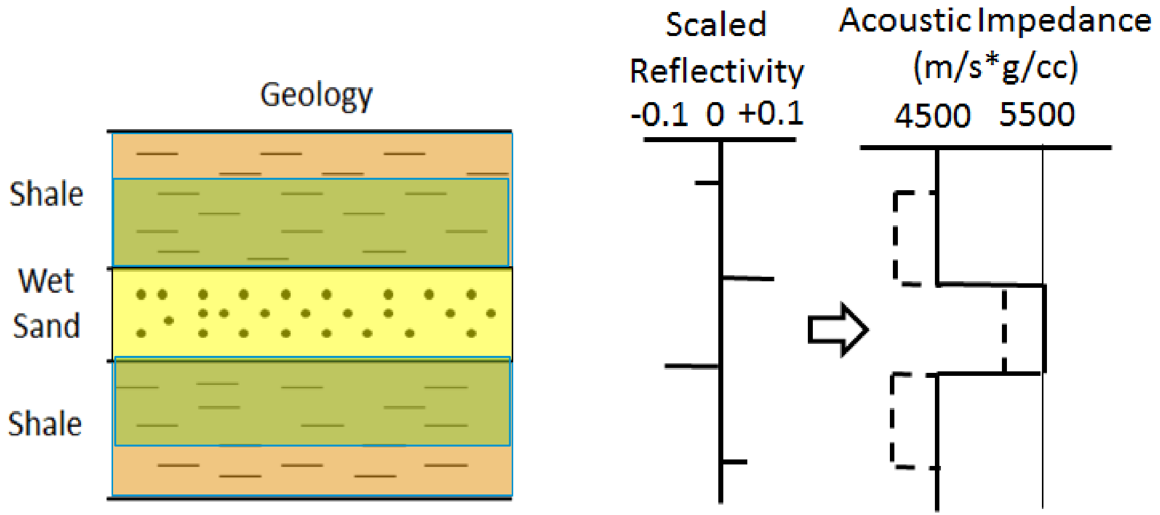
- This gives $w_0 = 0$ since the seismic and reflectivity both have zero mean.
- The second weight, $w_1 = 0.6$, is simply a scaling coefficient which matches the amplitudes between the seismic and reflectivity.

Linear regression

- Applying the regression coefficients gives:
- This can be recursively inverted to give:

$$\hat{\boldsymbol{r}} = w_1\boldsymbol{s} = 0.6\begin{bmatrix} -0.05 \\ +0.15 \\ -0.15 \\ +0.05 \end{bmatrix} = \begin{bmatrix} -0.03 \\ +0.09 \\ -0.09 \\ +0.03 \end{bmatrix}$$

$$\hat{I} = \begin{bmatrix} 4500 \\ 4238 \\ 5076 \\ 4238 \\ 4500 \end{bmatrix}$$



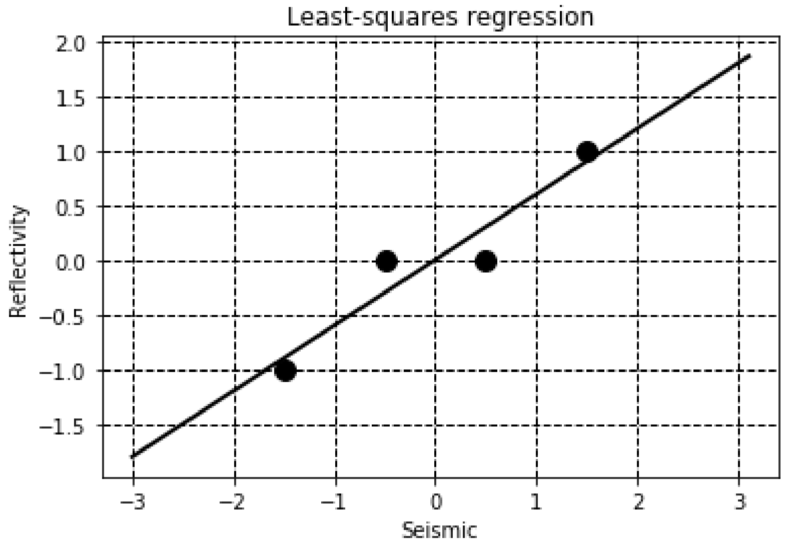
- For the sand layer, the result is worse than before regression!

Linear regression

- Another way to visualize the weights is to fit a straight line to the reflection coefficients versus seismic amplitudes, as shown here.
- The true values are shown by the black points and the line represents the equation:

$$\hat{\boldsymbol{r}} = 0 + 0.6\boldsymbol{s}$$

- In deconvolution we got a perfect fit because our model assumptions were correct.



- In least-squares regression, the points are fit in a “best” least-squares sense.

Steepest Descent

- In both the deconvolution and regression methods, we inverted the full matrix.
- For the large datasets used in seismic analysis this is impractical and we would normally use iterative techniques which do not involve calculating a matrix inverse.
- The simplest iterative technique is called gradient descent, or steepest descent (SD), in which we iteratively arrive at a solution by starting with an initial guess.
- The steepest descent algorithm for regression is written:

$$\boldsymbol{w}_{(k+1)} = \boldsymbol{w}_{(k)} + \alpha_{(k)}\boldsymbol{\delta}_{(k)}, \text{ where } k = 0,\ldots,K,$$

$$\boldsymbol{w}_{(k+1)} = \text{weights at } k+1^{st} \text{ iteration, } \alpha_{(k)} = \text{ learning rate, and } \boldsymbol{\delta}_{(k)} = (S^T S)\boldsymbol{w}_{(k)} - S^T \boldsymbol{r}.$$

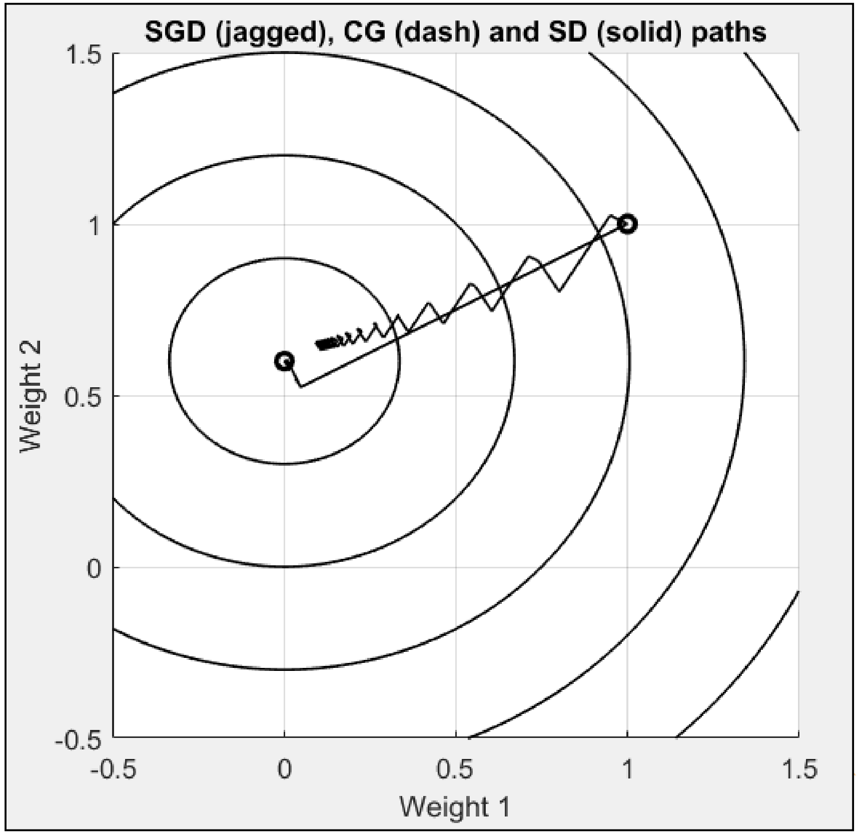
- Note that the gradient $\delta_{(k)}$ is the difference between the right and left sides of the equation used as a starting point for full least-squares inversion.

Conjugate Gradient and Stochastic Gradient Descent

- A more efficient iterative technique is the conjugate gradient (CG) algorithm, which takes steps which are orthogonal to the change in gradient.
- For linear problems, it can be shown that the CG algorithm always converges in the same number of steps as the number of unknown weights.
- A variant of the SD algorithm is called the least-mean-square, or LMS, algorithm which has applications in heart monitoring and noise cancelling headphones.
- In the LMS algorithm the weights are trained one sample at a time and thus the method is time-adaptive.
- In neural network applications, the LMS algorithm is called stochastic gradient descent (SGD).
- The figure on the next slide shows a comparison of all three algorithms applied to our regression problem.

Comparison of gradient descent methods

- Here, SGD is shown by the jagged line, CG by a dashed line and SD by a solid line.
- At this scale both CG and SD appear to take two steps, but at a larger scale we would see that SD actually takes several more steps.
- Each of the "jags" in the SGD algorithm represents an "epoch", where we cycle through the four samples.
- 10,000 epochs were used and the SGD algorithm still has not converged.
- The other two methods are in "batch" mode, with all the samples used simultaneously.

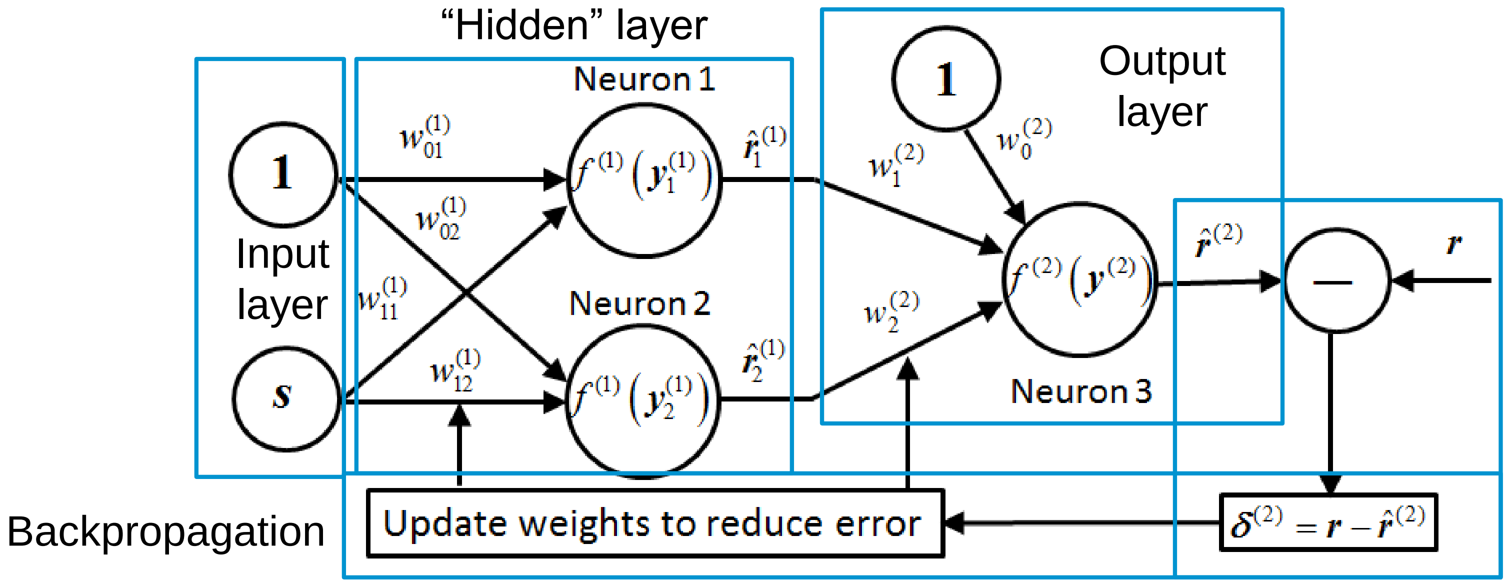


The feedforward neural network

- We saw that the straight-line solution given by linear regression did not give a perfect fit between the true seismic and reflectivity values.
- Neural networks, the oldest and best known type of machine learning algorithm, allow us to extend linear regression to nonlinear regression.
- The neural network we use has two different names which seem contradictory: the feedforward neural network and the backpropagation neural network.
- The term feedforward refers to how the output is computed from the input if the weights have already been determined.
- The term backpropagation refers to how the training of the weights is performed, using a technique called error backpropagation.
- Let’s now describe the algorithm and apply it to our problem.

The complete neural network

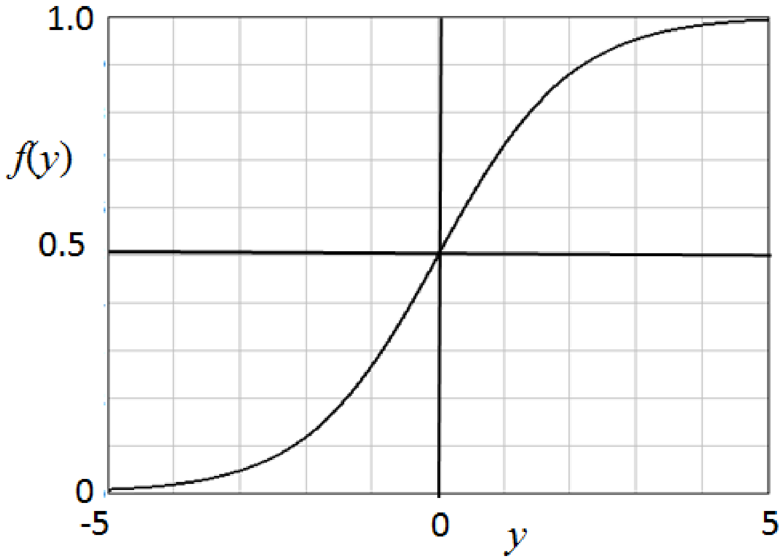
- Here is a diagram of the complete neural network we will use, consisting of an input layer, “hidden” layer, output layer and backpropagation algorithm:



- The key innovation in the network are the three neurons in the last two layers.

The logistic function

- The difference between linear and nonlinear regression is the neurons.
- Each neuron applies a linear or nonlinear function to the weighted inputs.
- The most common nonlinear function is the logistic (sigmoidal) function, shown here.
- The mathematical form of the logistic function is:



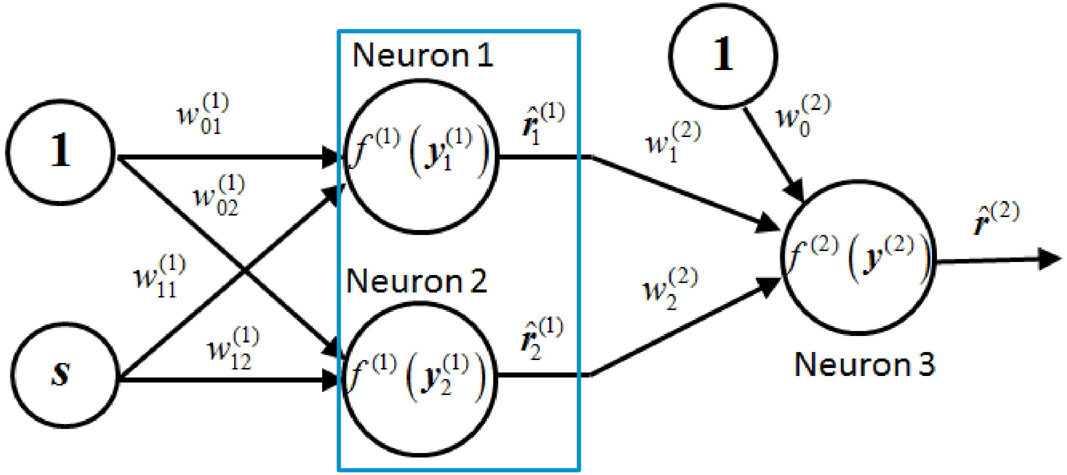
$$f(y) = \frac{1}{1+\exp(-y)}$$

- A computational advantage of the logistic function is its derivative, given by:

$$f'(y) = f(y)\left(1 - f(y)\right)$$

Applying the logistic function in the “hidden” layer

- The inputs are now weighted by four weights and fed into the first two neurons.
- Application of the logistic function to these weighted inputs gives two intermediate estimates of the reflectivity:



$$\hat{r}_1^{(1)} = \frac{1}{1+\exp\left(-y_1^{(1)}\right)},$$

$$\hat{r}_2^{(1)} = \frac{1}{1+\exp\left(-y_2^{(1)}\right)}.$$

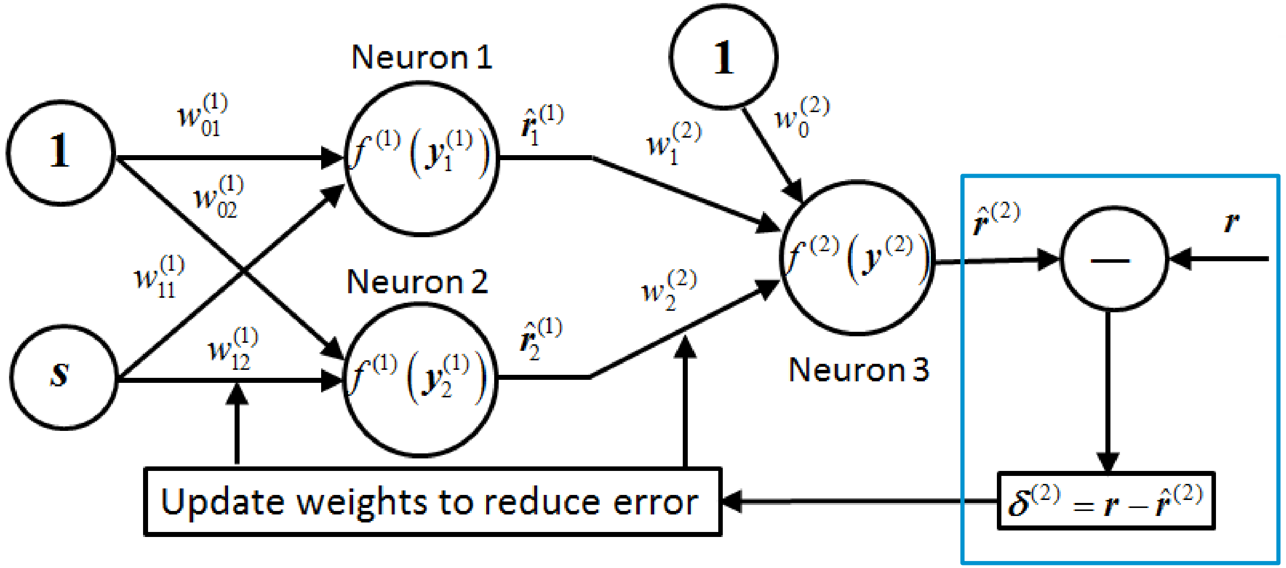
- This is called a “hidden” layer because the algorithm estimates the weights.
- Finally, these intermediate values are weighted and a final logistic function is applied:

$$\hat{r}_{ij}^{(1)} = \frac{1}{1+\exp\left(-y_{ij}^{(1)}\right)}$$

The backpropagation algorithm

- To find the optimum weights for our neural network a procedure called error backpropagation is used.
- The algorithm can be summarized as follows:
  - Initialize the weights in both layers to small random values.
  - Starting with the weights in the output layer, change the weights so as to minimize the error between the computed and desired output values.
  - Backpropagate the error minimization for all layers.
  - Iterate until an acceptable error is found.
- Let’s now look at error backpropagation in more detail.

The backpropagation algorithm



- The first step is to compute the error between the output of the network:

$$\delta^{(2)} = r - \hat{r}^{(2)}$$

- Then, this error is iteratively reduced using gradient descent (see appendix).

Applying the neural network

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

$$W_{(0)}^{(1)} = \begin{bmatrix} w_{01(0)}^{(1)} & w_{02(0)}^{(1)} \\ w_{11(0)}^{(1)} & w_{12(0)}^{(1)} \end{bmatrix} = \begin{bmatrix} 0.0940 & 0.4894 \\ -0.4074 & -0.6221 \end{bmatrix}, \text{ and } \boldsymbol{w}_{(0)}^{(2)} = \begin{bmatrix} w_{0(0)}^{(2)} \\ w_{1(0)}^{(2)} \\ w_{2(0)}^{(2)} \end{bmatrix} = \begin{bmatrix} 0.3736 \\ -0.633 \\ -0.263 \end{bmatrix}$$

- First, we scaled up the input and output values by a factor of 10 since neural networks work best with data normalized between -1 and 1.
- Applying the backpropagation technique with 10,000 iterations and $\alpha = 0.2$ produced the final weights given by:

$$W_{(10000)}^{(1)} = \begin{bmatrix} w_{01(10000)}^{(1)} & w_{02(10000)}^{(1)} \\ w_{11(10000)}^{(1)} & w_{12(10000)}^{(1)} \end{bmatrix} = \begin{bmatrix} -6.1001 & 6.0617 \\ -3.7842 & -3.8453 \end{bmatrix}, \text{ and } \boldsymbol{w}_{(10000)}^{(2)} = \begin{bmatrix} w_{0(10000)}^{(2)} \\ w_{1(10000)}^{(2)} \\ w_{2(10000)}^{(2)} \end{bmatrix} = \begin{bmatrix} 2.3384 \\ -2.5254 \\ -2.3382 \end{bmatrix}$$

Neural network results

- The computation of the final reflectivity is therefore given as:

$$\hat{r}_i^{(2)} = 0.1 * \left[2.3384 - \frac{2.5584}{1 + \exp\left(6.1001 + 3.7842 s_i\right)} - \frac{2.3382}{1 + \exp\left(-6.0617 + 3.8453 s_i\right)} \right]$$

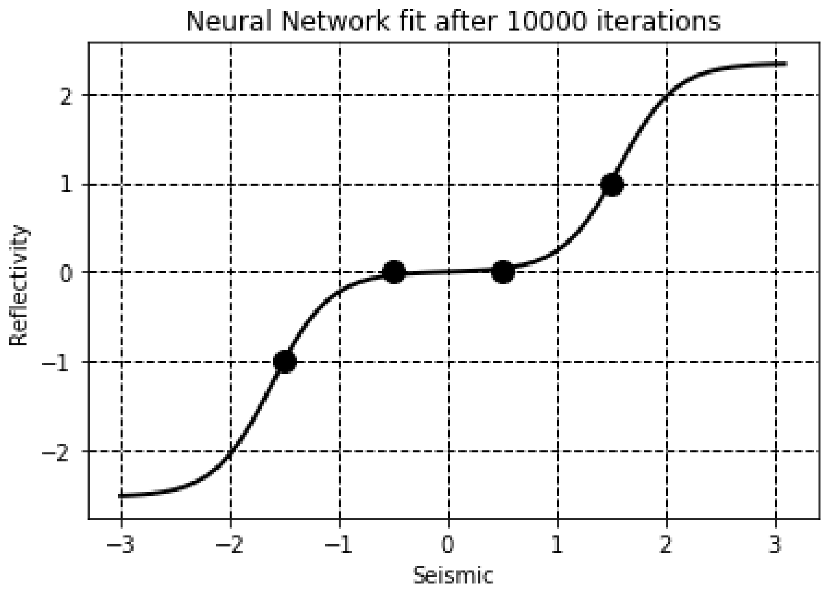
- The values of the final reflectivity are as follows:

$$\hat{\boldsymbol{r}}_{(10000)}^{(2)} = \begin{bmatrix} -0.0030 \\ +0.0999 \\ -0.0999 \\ +0.0030 \end{bmatrix}$$

- This is an almost perfect result and can be made as close to the correct answer as we want by increasing the number of iterations.

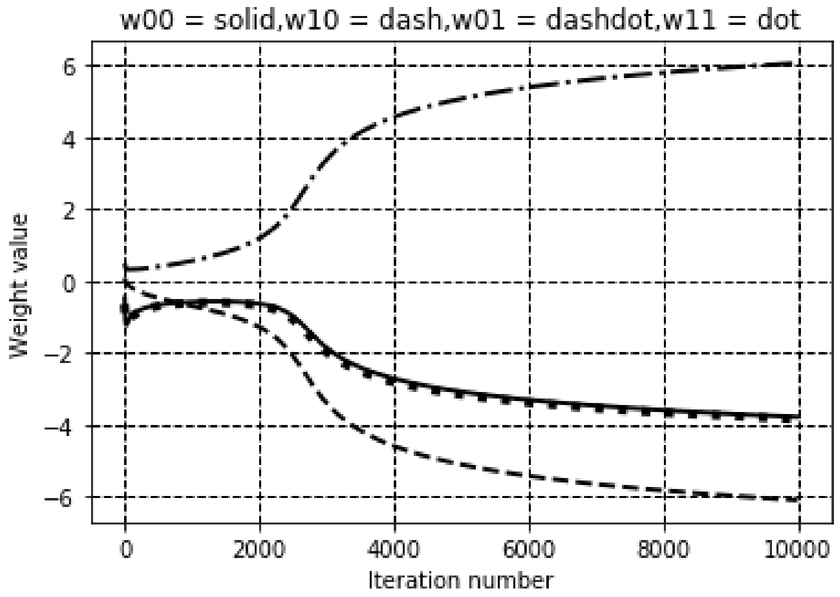
Neural network results

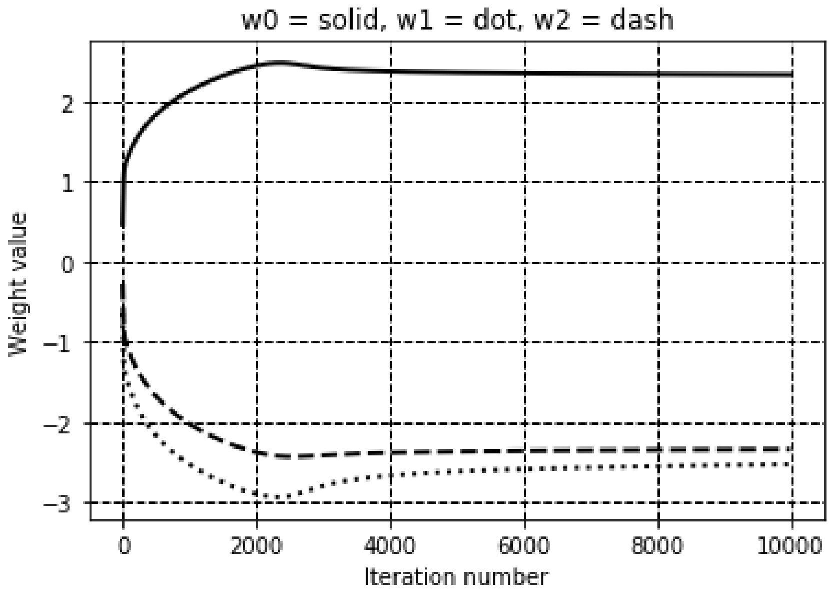
- As with the regression result, we can crossplot the output reflectivity against the input seismic.
- The result is shown on the right.
- Note the almost perfect fit at the input and output points, but the strong “imprint” of the logistic function.



- The obvious question is whether this mathematical transform has any relationship to the physics of the problem.
- Now let’s look at the weights themselves.

The weights after each iteration





- The weights as a function of iteration # for the four first-layer weights.
- Note the change after iteration 2000.

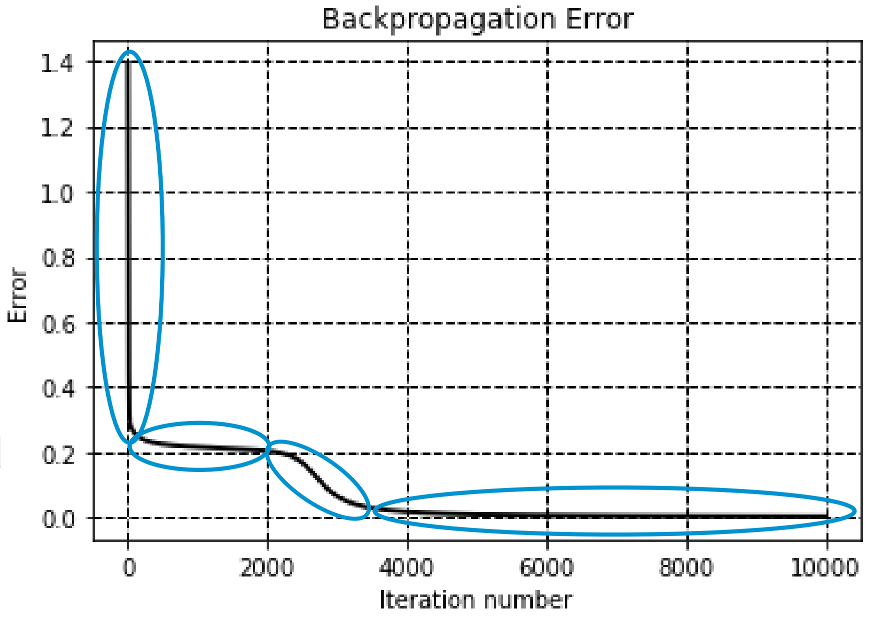
- The weights as a function of iteration # for the three second-layer weights.
- Note the change after iteration 2000.

The least-squared error

- Here is the least-squared error after each iteration, computed by the formula:

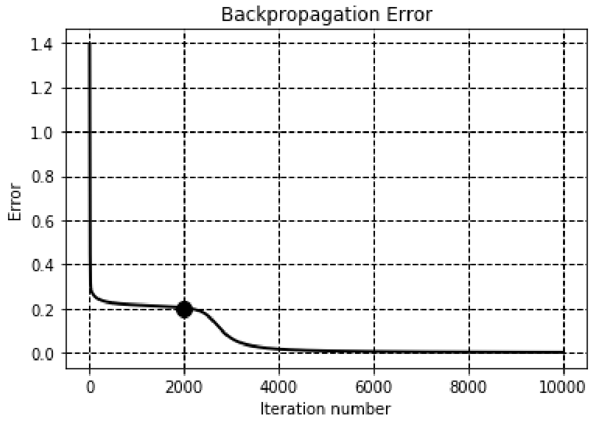
$$E_{(k)} = \frac{1}{2}\sum_{i=1}^{4}(r_i - \hat{r}_{i(k)}^{(2)})^2$$

- The error can be divided into four regions:
  - From iteration 1 to iteration 10 there is a dramatic drop in the error.
  - Between iterations 10 and 2000, the change in the error is almost flat, indicating we are trapped in a “local minimum”.
  - Between iterations 2000 and 3000 there is another sharp decrease in the error.
  - After iteration 3000 there is a gradual decline in the error towards zero.

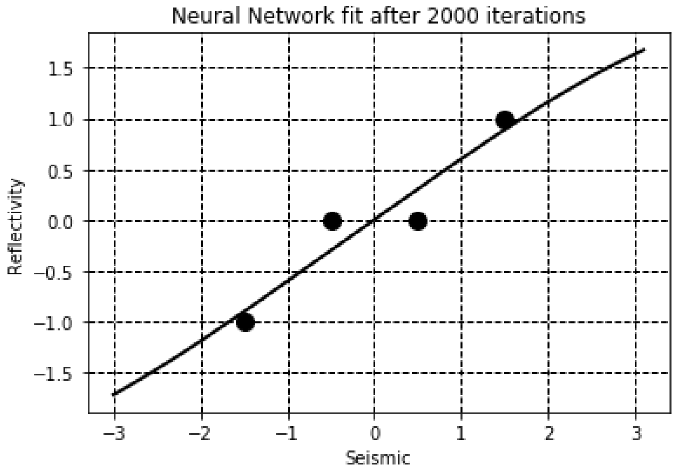


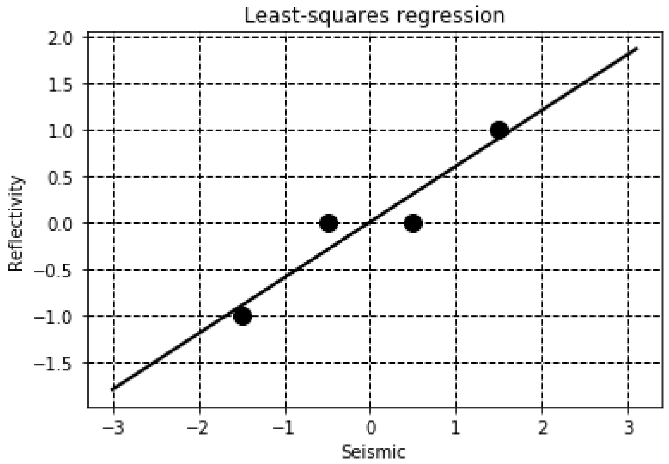
The local minimum

- Below, the least-squared error of the linear regression has been plotted at iteration 2000:

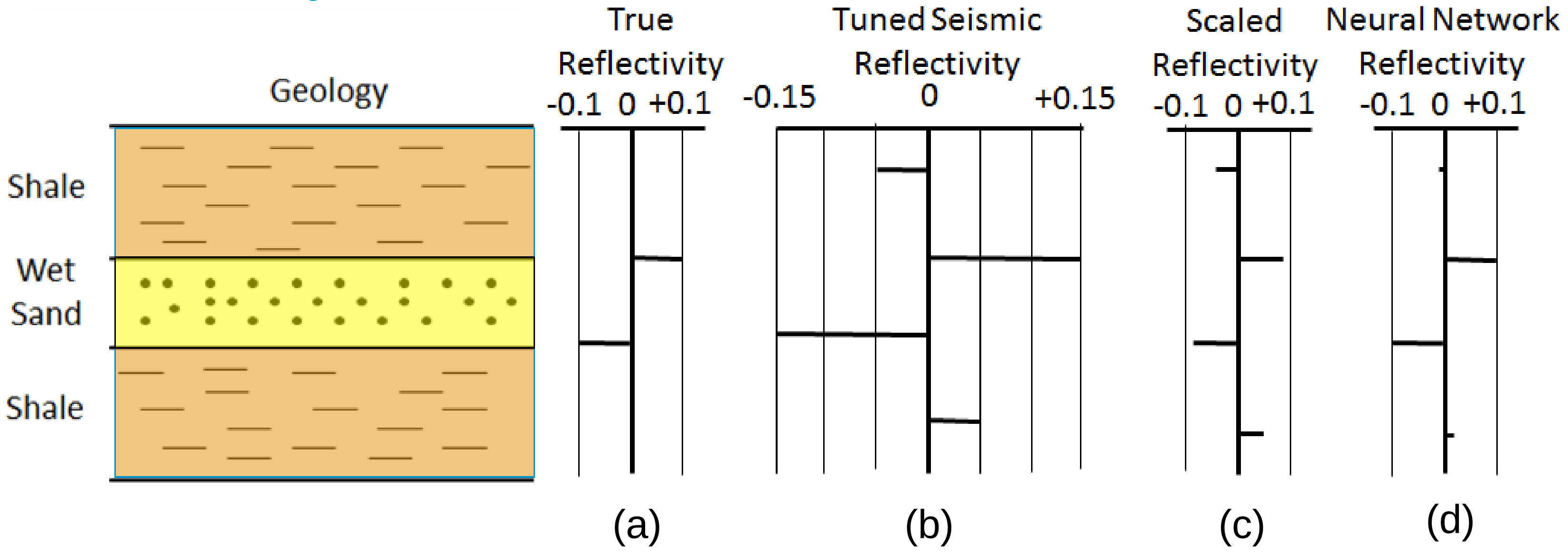


- The perfect fit suggests that the local minimum is close to the least-squares regression.
- This is confirmed by comparing the two plots at the right.





Summary of our simple model results

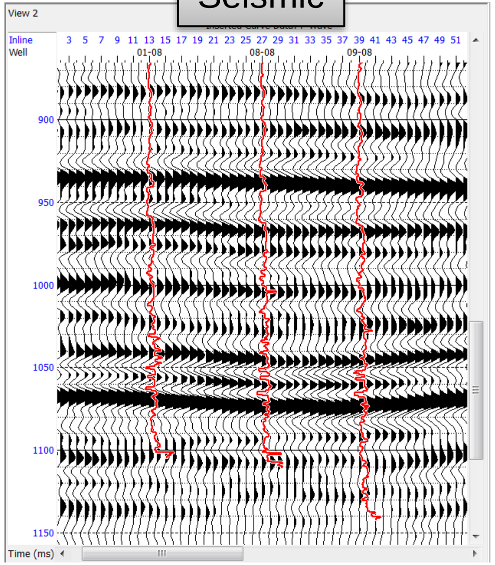


(a) shows the true reflectivity and the result of a "perfect" deconvolution,
(b) shows the convolution of a symmetrical wavelet with the reflectivity,
(c) shows the least-squared scaling of (b) to match (a), and
(d) shows the neural network prediction of (a) from (b).

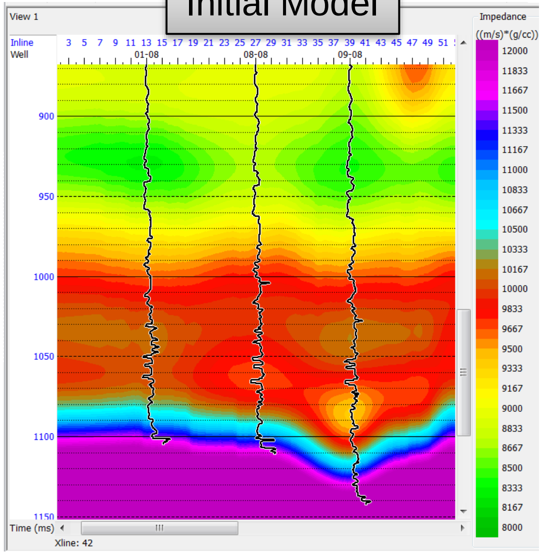
Seismic Inversion on real data

- Our first example was a simple numerical example that we could do by hand.
- Normally, inversion is done on multiple CPUs or GPUs, where the input is on the left, a low frequency model in the center and the impedance inversion on the right:

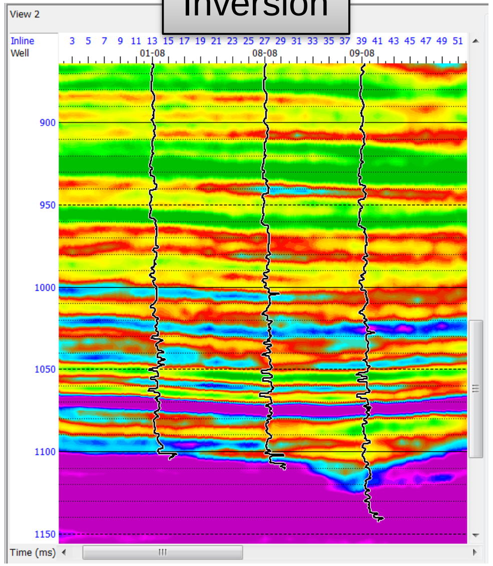
Seismic



Initial Model

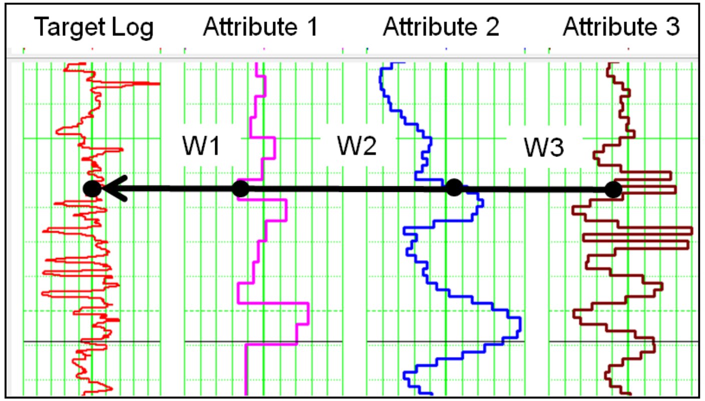


Inversion



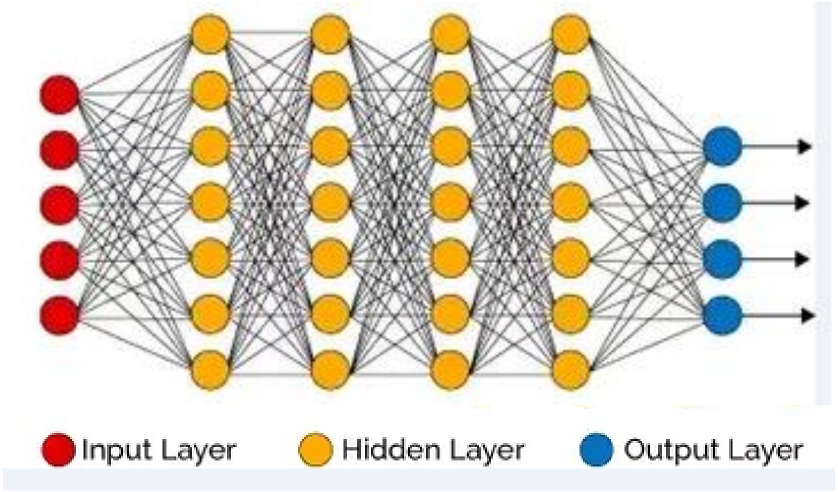
Supervised Learning (Emerge)

- Hampson et al. (2001) described a supervised learning methodology to predict log properties like impedance using the following flow:
  - The input of multiple attributes generated from the seismic data.
  - The development of a statistical relationship by analyzing a set of training data at well locations.
  - The use of either a linear (multivariate regression) or nonlinear (single hidden layer neural network).
  - The use of cross-validation to estimate the reliability of the derived relationship.



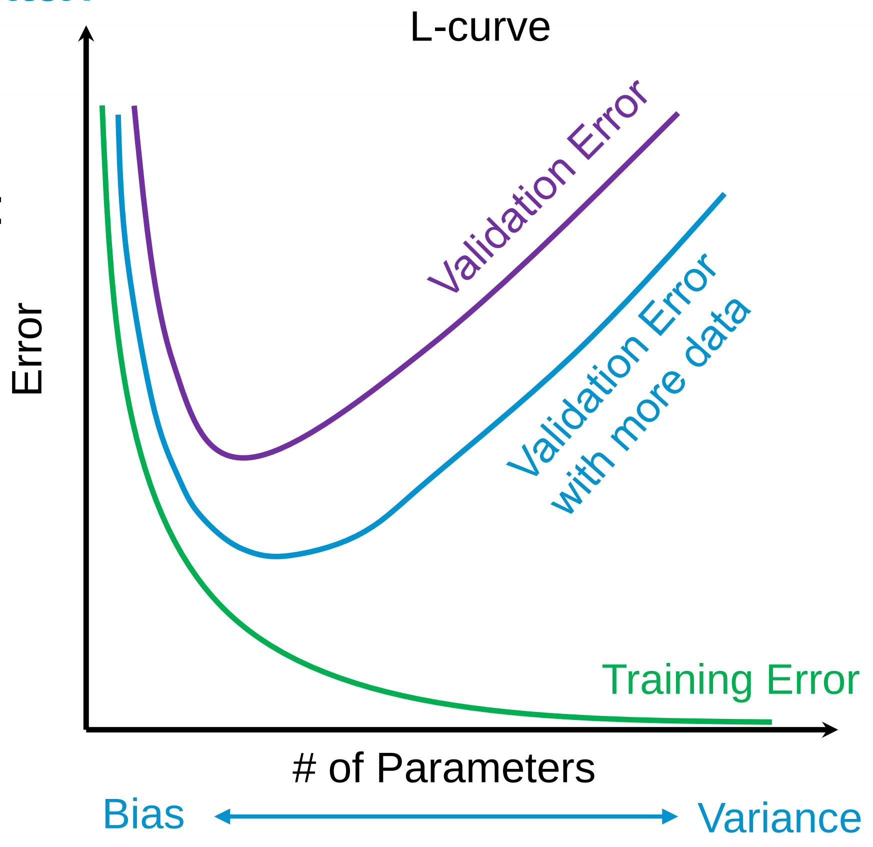
Deep Neural Networks (DNNs)

- We have updated that inversion flow using a Deep Neural Network (DNN) with many hidden layers.
- If a neural network has many hidden layers it can model complex nonlinear relationships.
- The weights are solved as large nonlinear inverse problem using iterative techniques.
  - The solution for the weights is non-unique.
- Similar to linear methods:
  - The weights are calculated on training data.
  - To ensure the network is not over trained the network is tested on a validation data set.



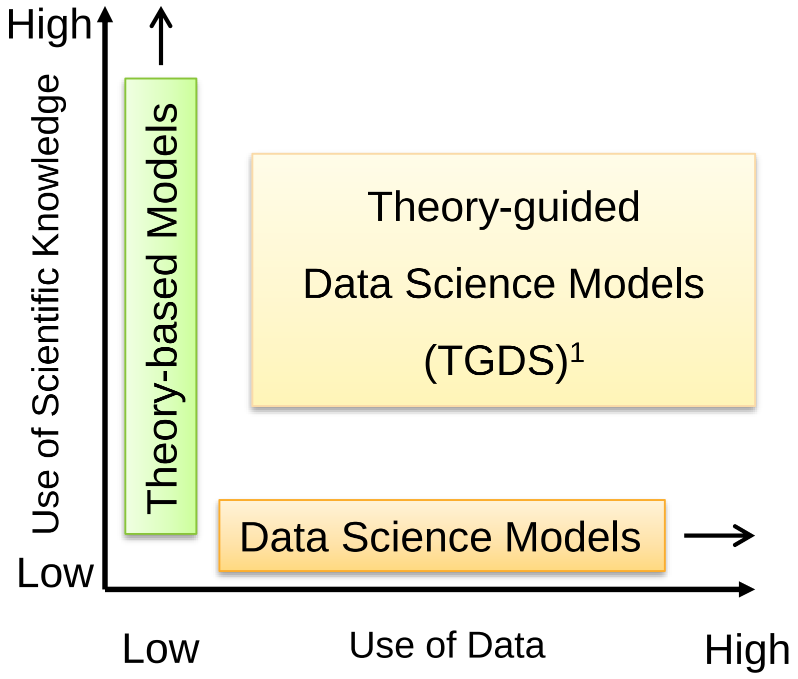
Do we have enough training data?

- Deep neural networks have many layers and parameters, which increases the risk of overfitting, where:
  - Overfitting is characterized by observing:
    - Small training error
    - Large validation error
- Possible solutions
  - Reduce the number of parameters / layers
  - Regularization, early stopping
  - Increase the amount data:
    - Needs to be labelled data!
    - Synthetic data
  - Theory-guided data science



Theory-guided data science models (TGDS)

- A recent paper by Karpatne et al (2017) suggests a new approach to scientific discovery, which combines both theory and machine learning.
- Traditional theory-based models make high use of scientific knowledge.
- However, the newer data science-based models make high use of data.
- Theory-guided data science (TDGS) modelling combines the best of both.
- But this approach requires a large number of representative samples.

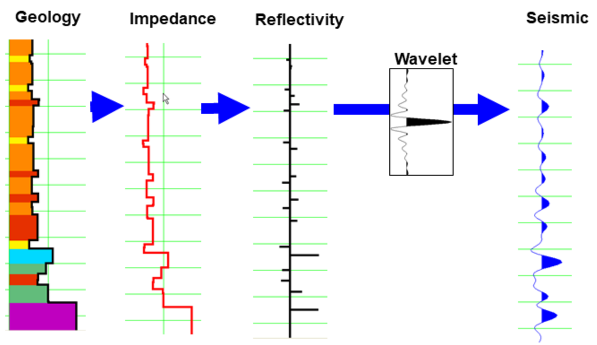


[1]Karpatne et al., 2017, "**Theory-guided data science: A New paradigm for scientific discovery**"

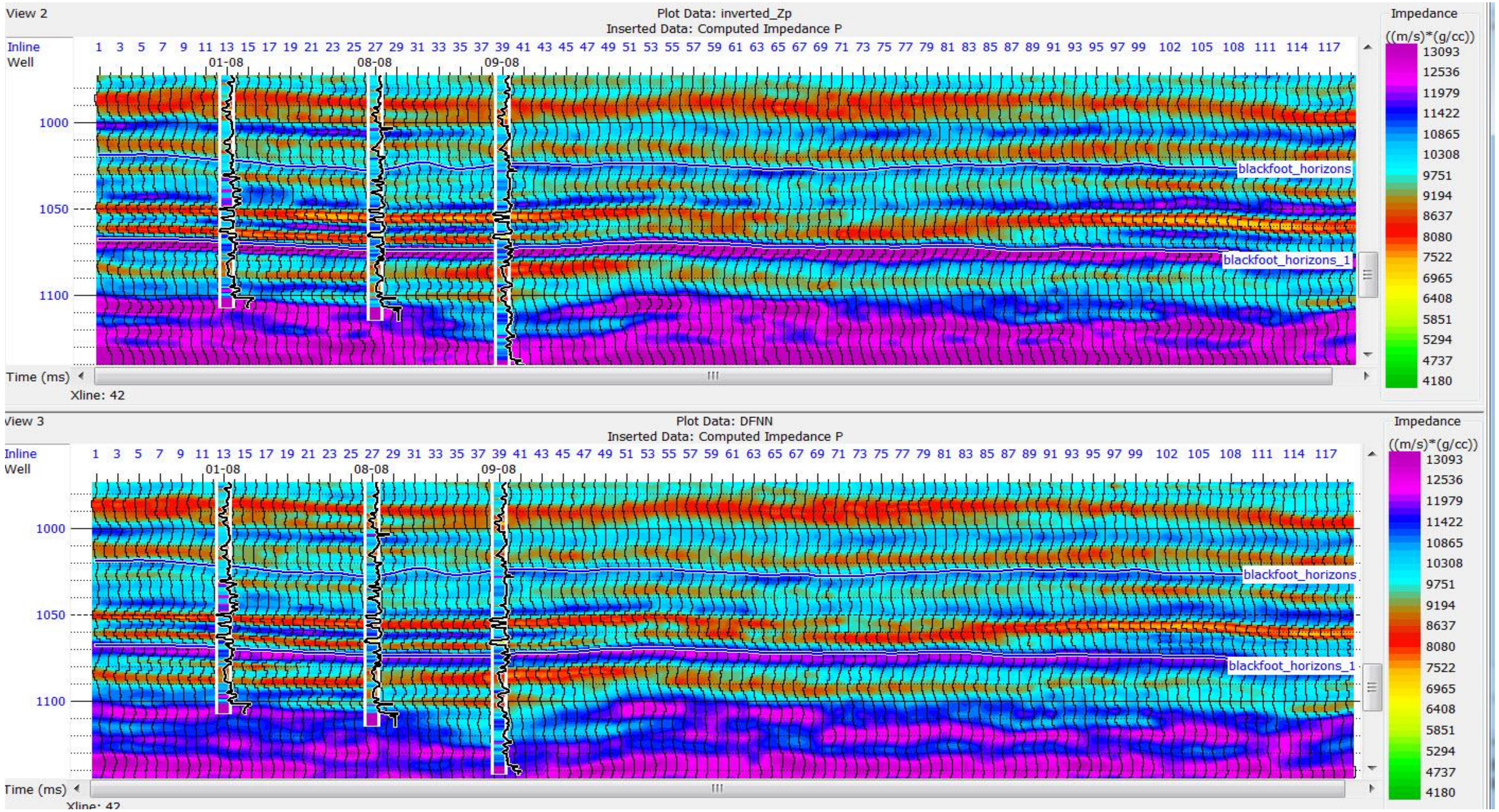
Machine Learning Post-stack Inversion

- Following Karpatne et al. (2017) our Machine Learning Inversion uses a TGDS model, where:
  - We build a 2D impedance model from well control, using 12 wells in all.
  - We generate post-stack synthetics for each location using a wavelet derived from the seismic.
  - The outputs of the theory-based component are then used as inputs in the data science component.
- That is, the synthetic data is used to train and validate a DNN.
- The trained DNN is then applied to the real data.

Theory-based model

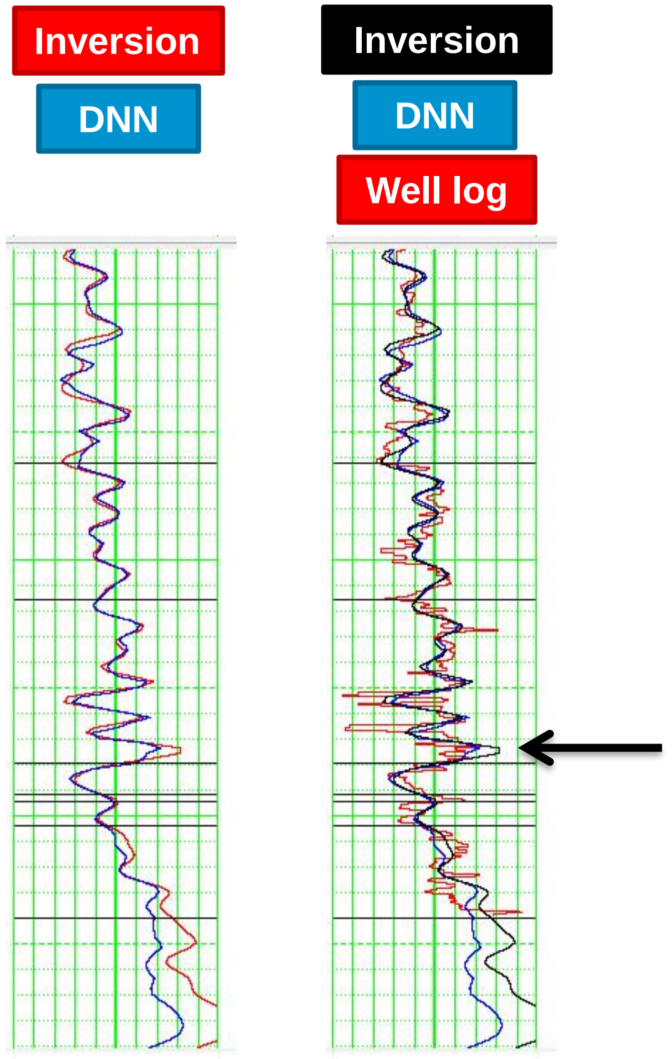


Conventional inversion (top) Vs. DNN (bottom)

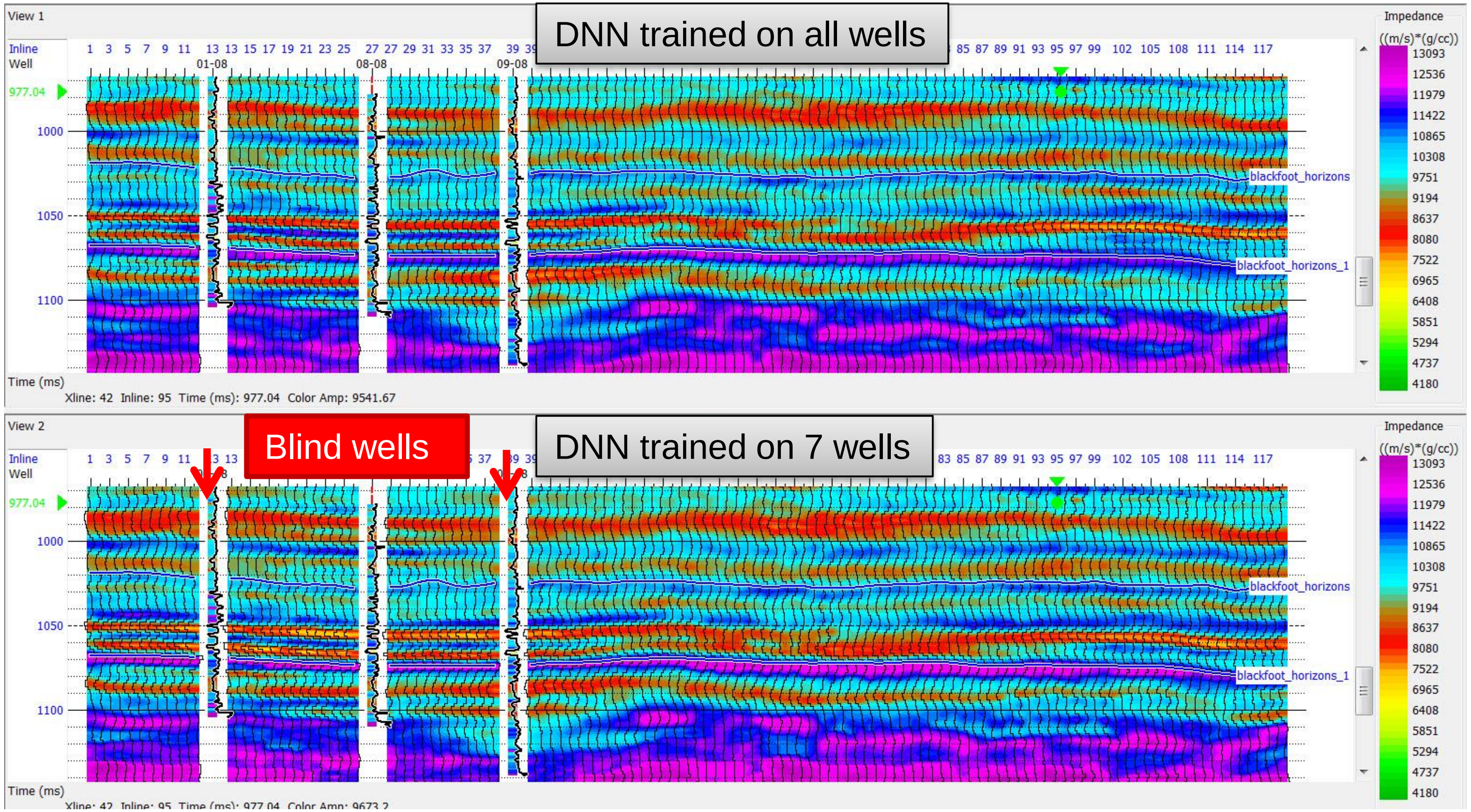


Comparison at the 09-08 well

- The results look amazingly similar.
- Note that the character of the results is very similar, with slight differences in amplitudes.
- Where there is a noticeable difference, (arrow), the DNN matches the log curve better.



Blind locations



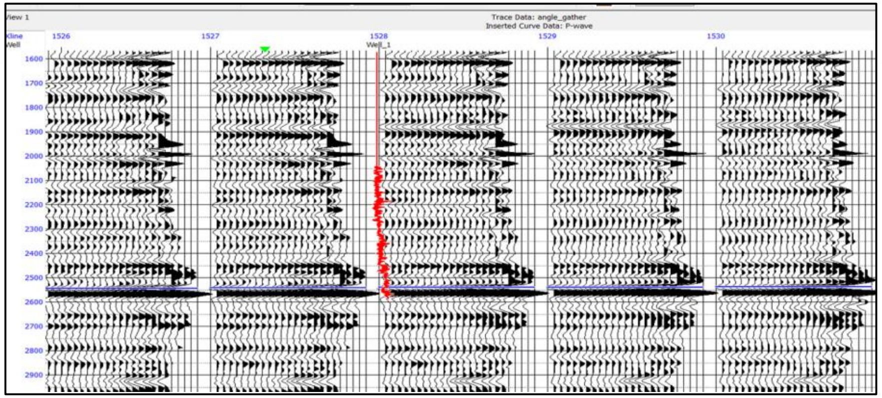
Xline: 42 Inline: 95 Time (ms): 977.04 Color Amp: 9541.67

Xline: 42 Inline: 95 Time (ms): 977.04 Color Amp: 9673.2

Gulf Coast example

- We then extended our algorithm to pre-stack data where we use a hybrid theory and data model to predict reservoir properties:
  - Rock physics relationships were used to simulate a large, idealized set of well logs and synthetics.
  - Pre-stack synthetic data was used to train the DNN to estimate elastic and rock properties.
- Unlike our previous example, only one well is used in this analysis.

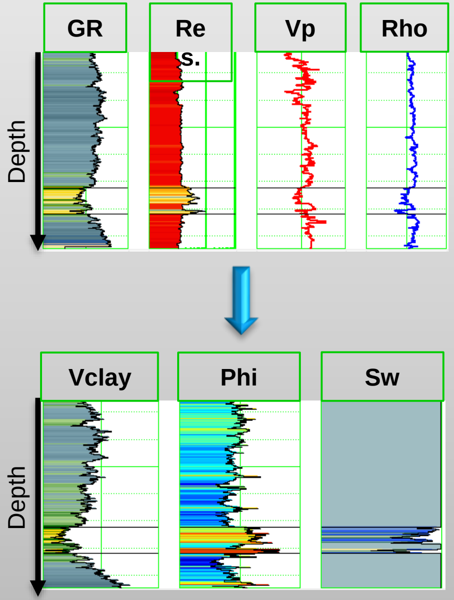
Angle Gathers



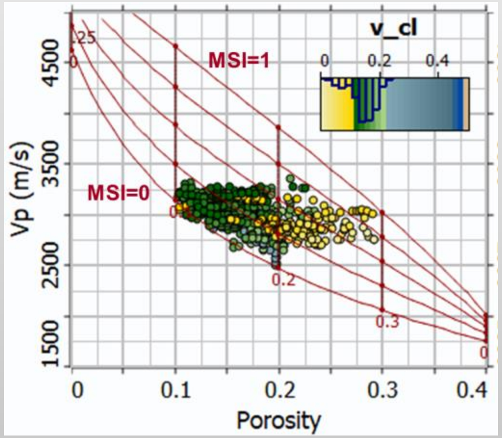
- The seismic data above was processed in a manner suitable for simultaneous inversion.
- Our flow is shown in the next slides.

Synthetic Catalog Workflow

Step 1: Petrophysical Analysis



Step 2: Rock Physics Model Calibration



Rock Physics Template of the **unconsolidated (soft) sand model** extended to the **intermediate** and **stiff sand models** through the **Matrix Stiffness Index (MSI) parameter (Allo, 2019)**.

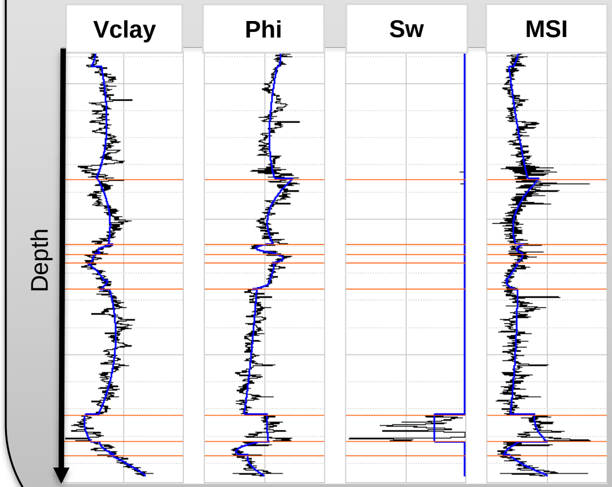
Establish the rock physics model

$Vp, Vs, \rho = \mathrm{RPM}(\phi, V_{cl}, S_w, MSI)$

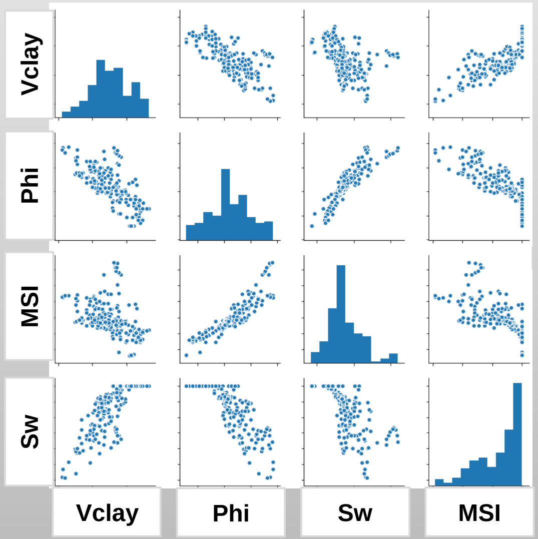
Synthetic Catalog Workflow

Step 3: Statistical analysis

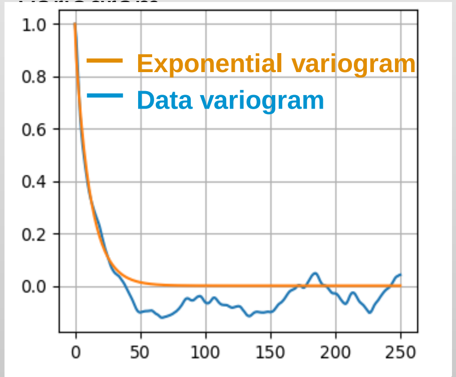
a) Define **lithofacies** and calculate the **background trend**



b) Establish the statistics for each lithofacies: the covariance matrix



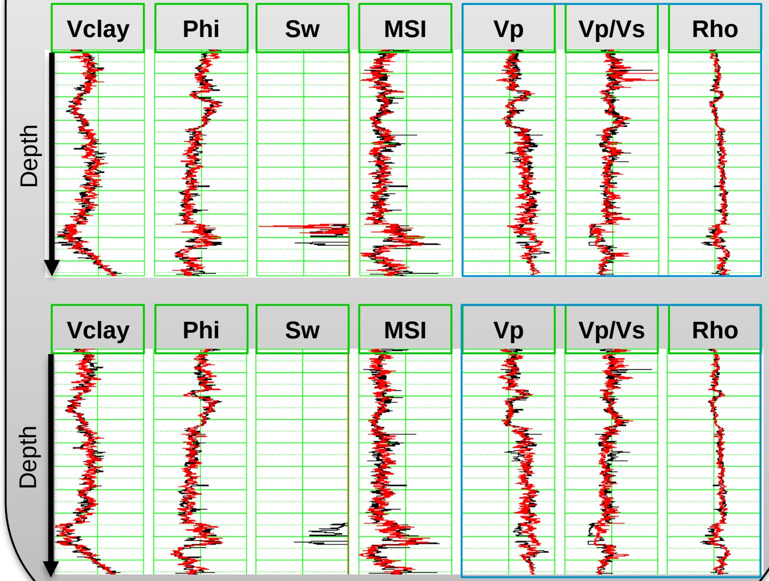
c) Model the vertical continuity: the spatial



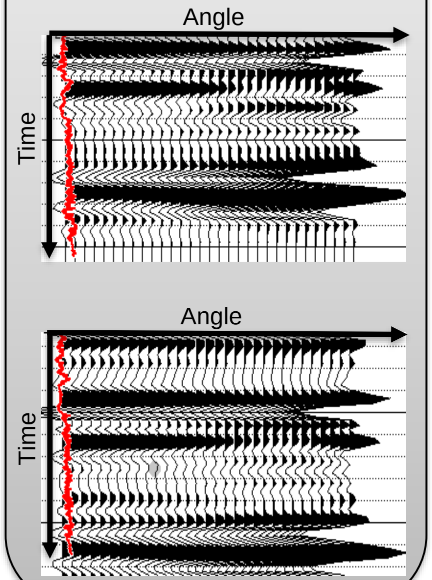
The variogram influences the vertical resolution of the simulations generated in next step.

Synthetic Catalog Workflow

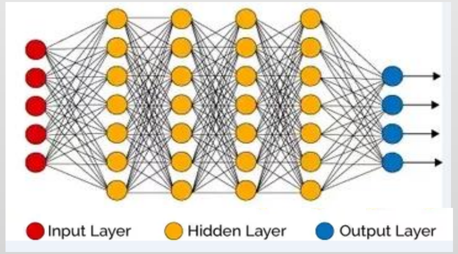
Step 4: Elastic Properties Simulations

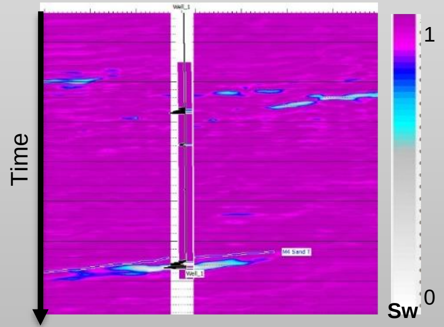


Step 5: Synthetics generation



Step 6: DFNN training and application

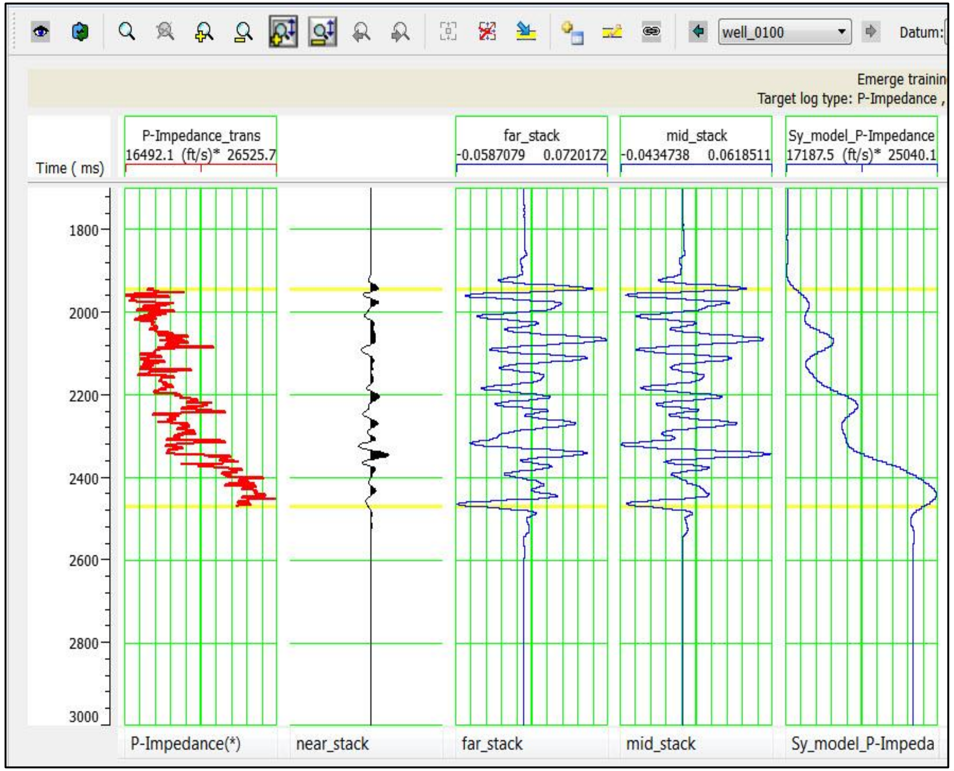




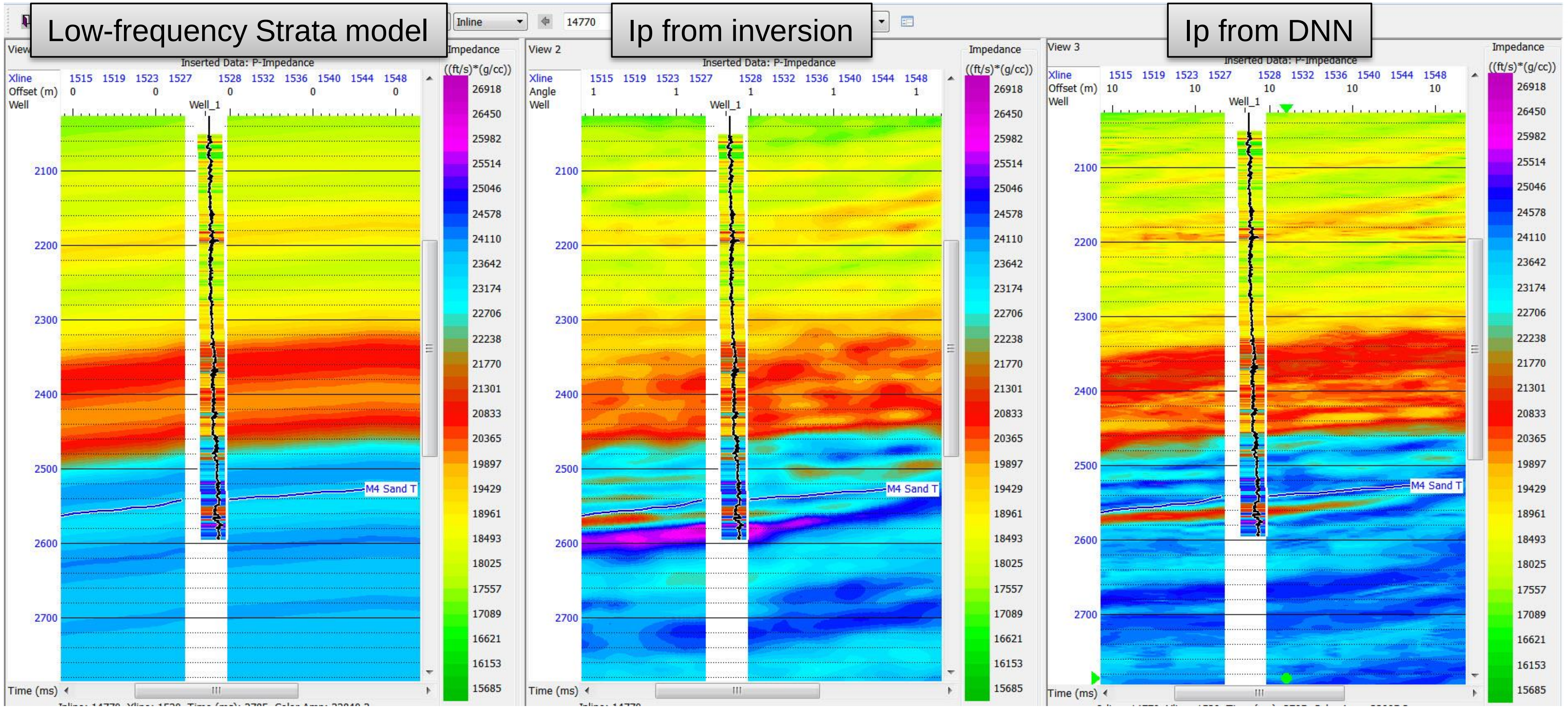
Saturation prediction

Training and validating the DNN operator

- Any log curve can be specified as the target. In the example the **P-wave impedance** is the Target log.
- The input attributes are calculated from the near, mid and far angle stacks calculated from the synthetic gathers.
- In addition, the low-frequency P-wave impedance strata model is input.

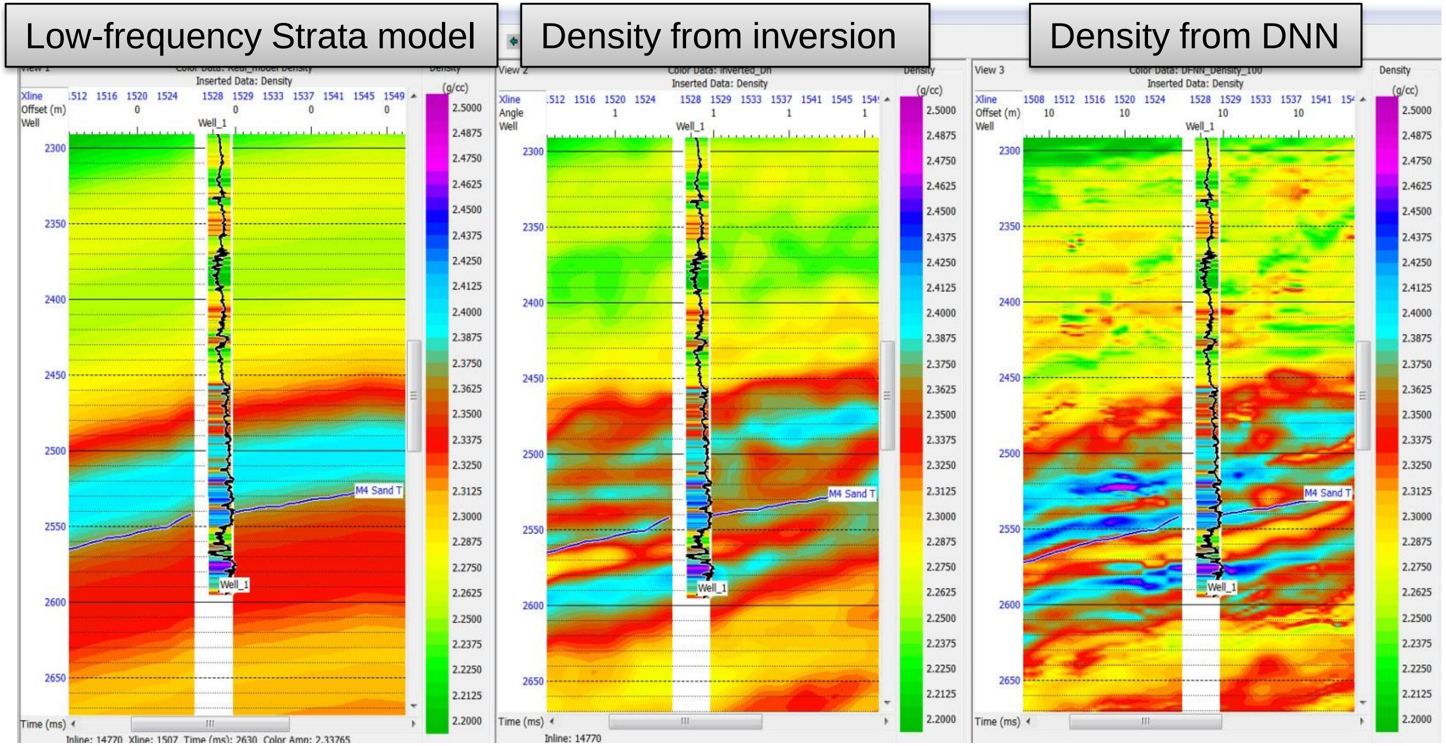


Applying the DNN operator to the real data

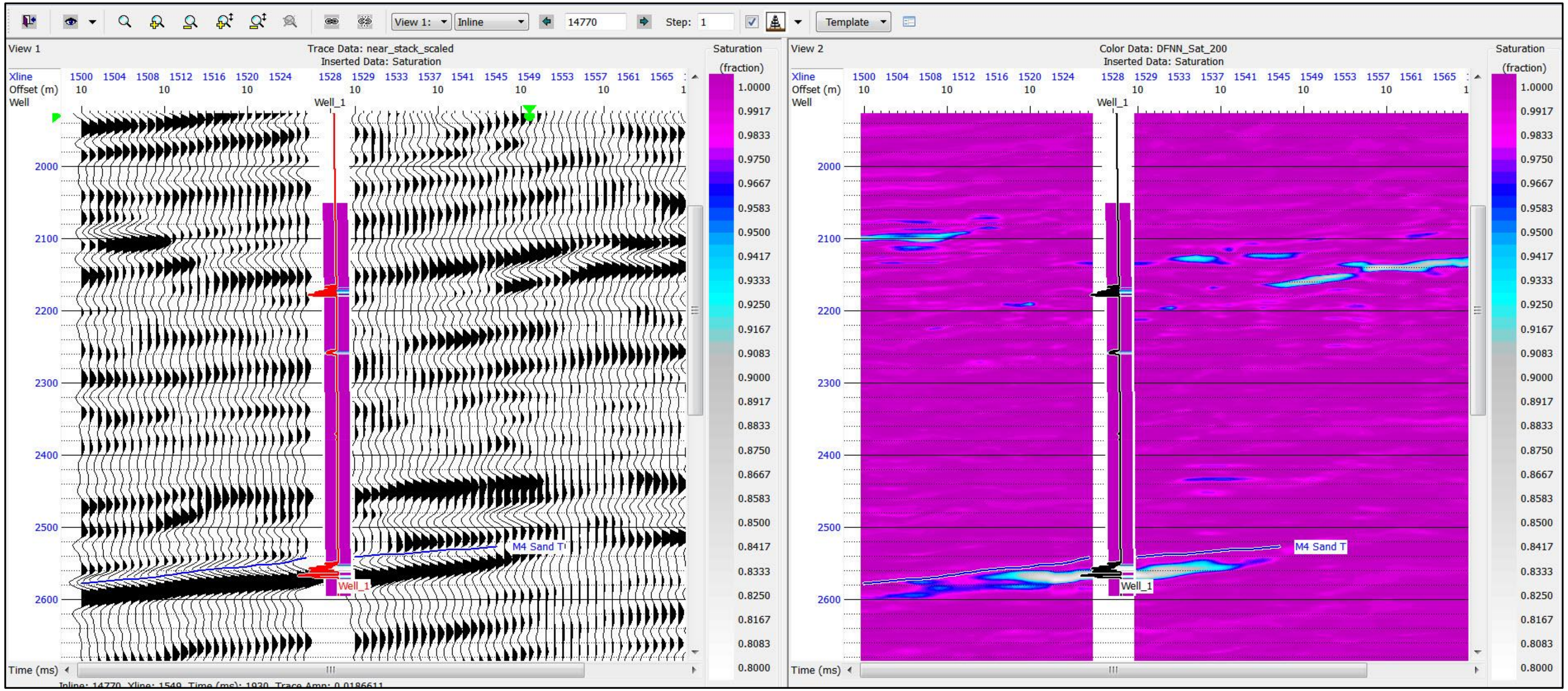


Applying the DNN operator to the real data

The density predicted by DNN gives a higher resolution result than pre-stack inversion and appears to tie the well better.



Gas Saturation

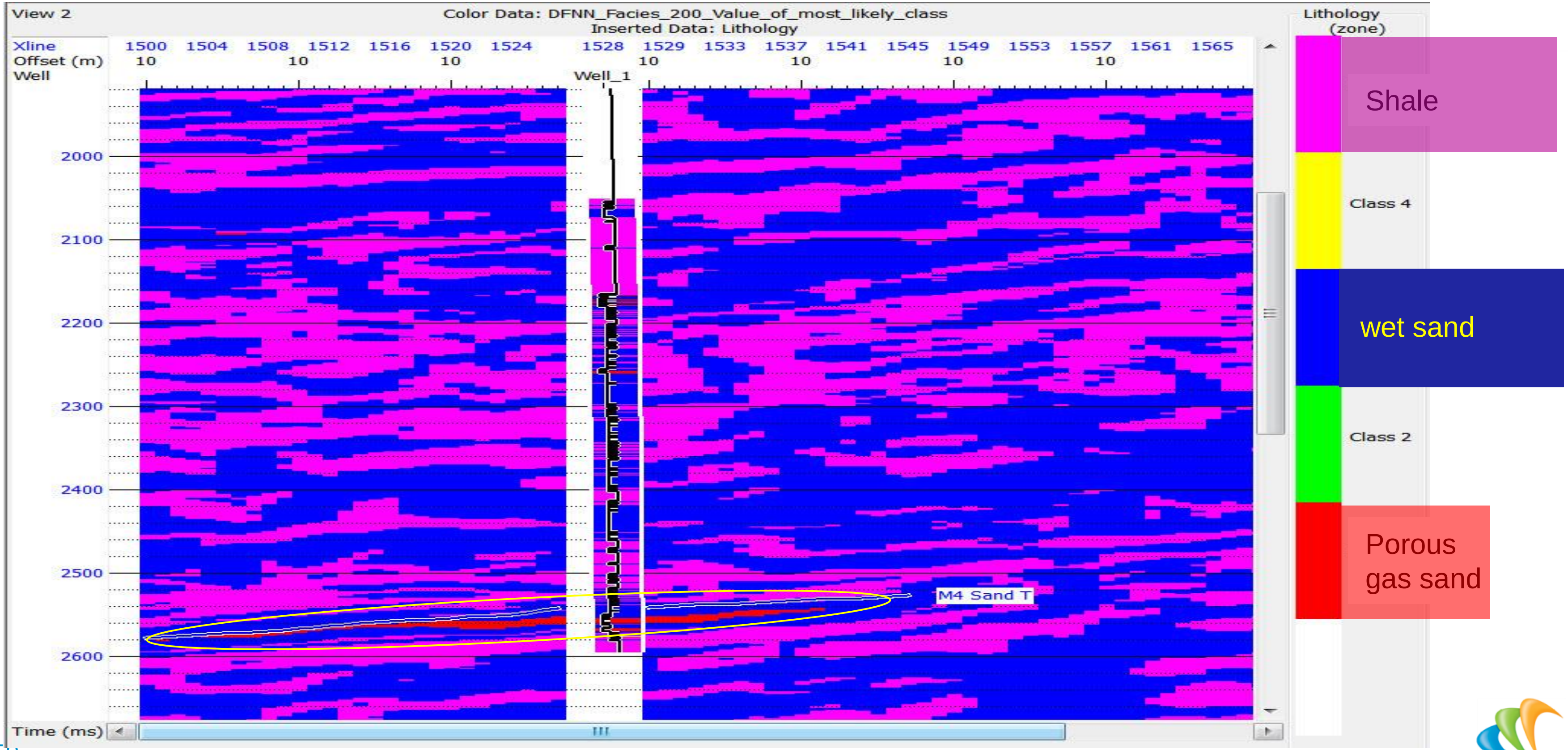

View 1: Inline 14770 Step: 1 Template

View 1 — Trace Data: near_stack_scaled, Inserted Data: Saturation

View 2 — Color Data: DFNN_Sat_200, Inserted Data: Saturation

Inline: 14770 Xline: 1549 Time (ms): 1930 Trace Amp: 0.0186611

Application of DNN lithology prediction to real volume



View 2
Color Data: DFNN_Facies_200_Value_of_most_likely_class
Inserted Data: Lithology
Xline 1500 1504 1508 1512 1516 1520 1524 1528 1529 1533 1537 1541 1545 1549 1553 1557 1561 1565
Offset (m) 10 10 10 10 10 10 10
Well
Well_1
2000
2100
2200
2300
2400
2500
2600
Time (ms)
M4 Sand T
Lithology (zone)
Shale
Class 4
wet sand
Class 2
Porous gas sand

Summary

- In the first part of the talk, I gave a detailed comparison between conventional and machine learning inversion using a simple example.
- I then moved to a hybrid theory-guided data science (TGDS) model approach for inverting large datasets, in which:
  - Rock physics and seismic theory is used to generate synthetic data which is then used to train the neural network.
  - The theory is then used to generate data not present in the well data.
- Both post-stack and pre-stack inversion examples were shown
  - The post-stack results were nearly identical.
  - The machine learning pre-stack inversion had higher frequencies than the theory based method.
  - The DNN allows for nonlinear models so we can estimate target variables such as fluid saturation or lithology.

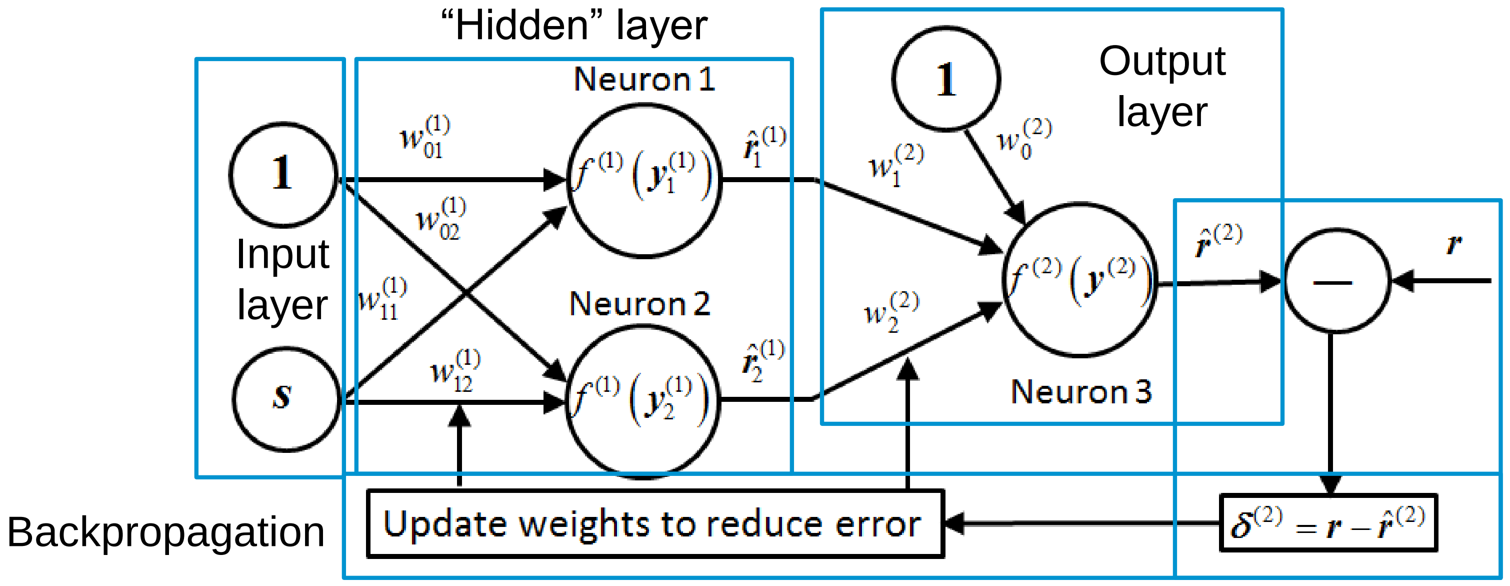
Questions?

cgg.com

Passion for Geoscience

Appendix: The complete neural network solution

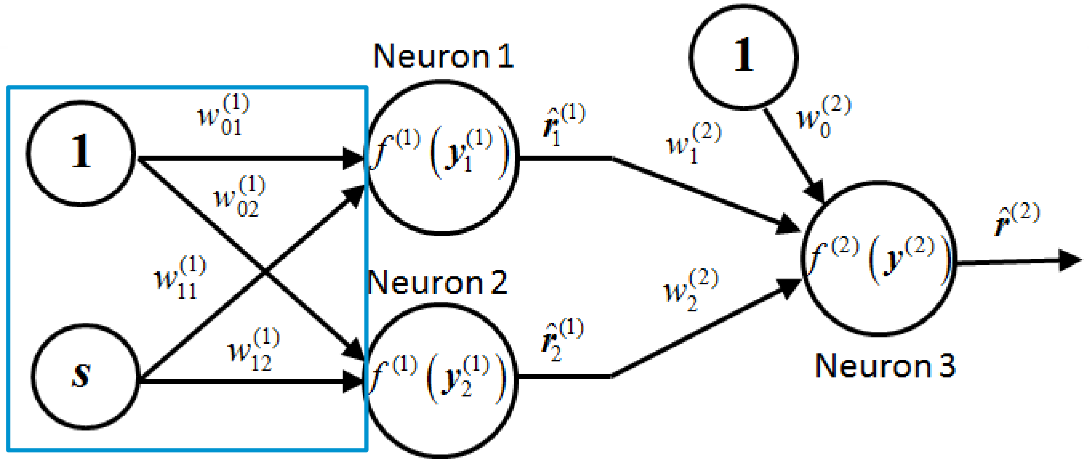
- Here is a diagram of the complete neural network we will use, consisting of an input layer, “hidden” layer, output layer and backpropagation algorithm:



- The key innovation in the network are the three neurons in the last two layers.

The feedforward neural network

- In the first part of the process we apply two sets of bias and gradient weights to the seismic samples.
- This can be written in vector or matrix format as follows, where superscript (1) is the first layer:



Neuron 1
Neuron 2
Neuron 3
1
s
1
$w_{01}^{(1)}$
$w_{02}^{(1)}$
$w_{11}^{(1)}$
$w_{12}^{(1)}$
$f^{(1)}\left(\boldsymbol{y}_1^{(1)}\right)$
$f^{(1)}\left(\boldsymbol{y}_2^{(1)}\right)$
$\hat{\boldsymbol{r}}_1^{(1)}$
$\hat{\boldsymbol{r}}_2^{(1)}$
$w_1^{(2)}$
$w_2^{(2)}$
$w_0^{(2)}$
$f^{(2)}\left(\boldsymbol{y}^{(2)}\right)$
$\hat{\boldsymbol{r}}^{(2)}$

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

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

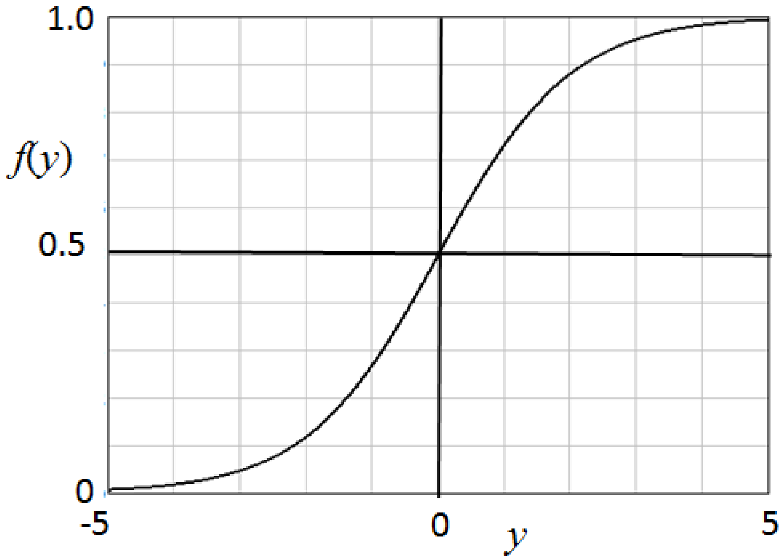
where $\mathbf{1}^T = \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}$, and

$$\boldsymbol{s}^T = \begin{bmatrix}-.05 & .15 & -.15 & .05\end{bmatrix}$$

$$\Rightarrow Y^{(1)} = \begin{bmatrix}\boldsymbol{y}_1^{(1)} & \boldsymbol{y}_2^{(1)}\end{bmatrix} = SW^{(1)} = \begin{bmatrix}1 & -0.05 \\ 1 & +0.15 \\ 1 & -0.15 \\ 1 & +0.05\end{bmatrix}\begin{bmatrix}w_{01}^{(1)} & w_{02}^{(1)} \\ w_{11}^{(1)} & w_{12}^{(1)}\end{bmatrix}$$

The logistic function

- The difference between linear and nonlinear regression is the neurons.
- Each neuron applies a linear or nonlinear function to the weighted inputs.
- The most common nonlinear function is the logistic (sigmoidal) function, shown here.
- The mathematical form of the logistic function is:



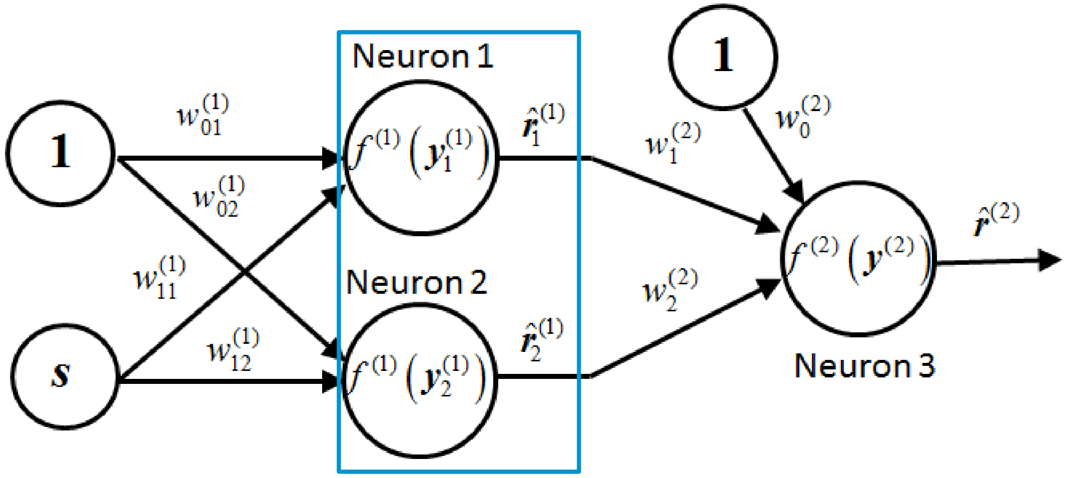
$$f(y) = \frac{1}{1+\exp(-y)}$$

- A computational advantage of the logistic function is its derivative, given by:

$$f'(y) = f(y)\left(1 - f(y)\right)$$

Applying the logistic function in the “hidden” layer

- Thus, we will next apply the logistic functions in the “hidden” layer.
- This gives two intermediate estimates of the reflectivity, which again can be written in vector or matrix format:



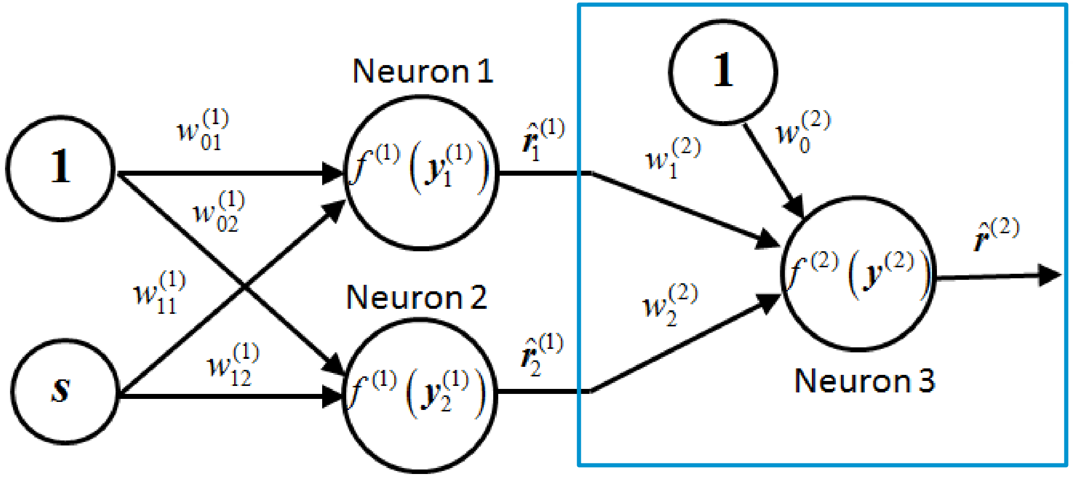
$$\hat{\boldsymbol{r}}_1^{(1)} = \frac{1}{1+\exp\left(-\boldsymbol{y}_1^{(1)}\right)},$$

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

$$\Rightarrow R^{(1)} = F^{(1)}\left(Y^{(1)}\right) = \begin{bmatrix} 1 & \hat{r}_{11}^{(1)} & \hat{r}_{12}^{(1)} \\ 1 & \hat{r}_{21}^{(1)} & \hat{r}_{22}^{(1)} \\ 1 & \hat{r}_{31}^{(1)} & \hat{r}_{32}^{(1)} \\ 1 & \hat{r}_{41}^{(1)} & \hat{r}_{42}^{(1)} \end{bmatrix}, \hat{r}_{ij}^{(1)} = \frac{1}{1+\exp\left(-y_{ij}^{(1)}\right)}.$$

The output layer

- Finally, we get to the output layer, shown by the superscript (2).
- This involves first computing a weighted sum of the intermediate wavelets with a new bias and two gradient weights:



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

$$\hat{\boldsymbol{r}}_1^{(1)T} = \begin{bmatrix} \hat{r}_{11}^{(1)} & \hat{r}_{21}^{(1)} & \hat{r}_{31}^{(1)} & \hat{r}_{41}^{(1)} \end{bmatrix}, \text{ and}$$

$$\hat{\boldsymbol{r}}_2^{(1)T} = \begin{bmatrix} \hat{r}_{12}^{(1)} & \hat{r}_{22}^{(1)} & \hat{r}_{32}^{(1)} & \hat{r}_{42}^{(1)} \end{bmatrix}.$$

- Our second function is linear, giving:

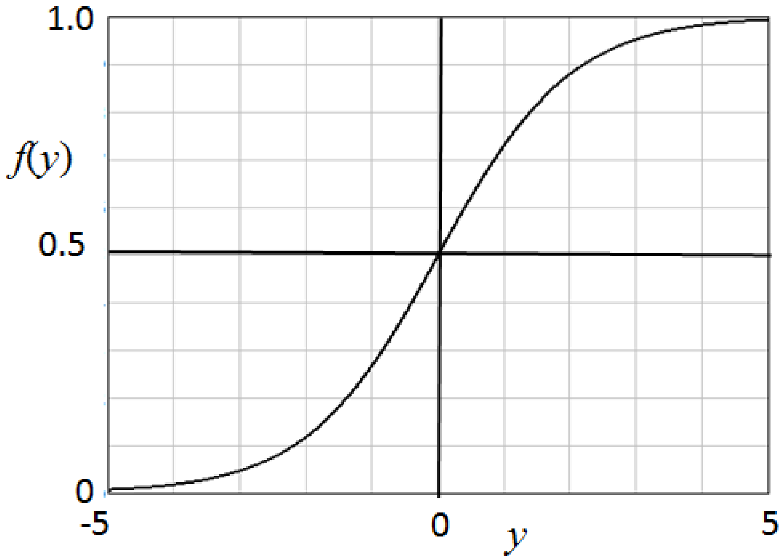
$$\hat{\boldsymbol{r}}^{(2)} = f^{(2)}\left(w_0^{(2)}\mathbf{1} + w_1^{(2)}\hat{\boldsymbol{r}}_1^{(1)} + w_2^{(2)}\hat{\boldsymbol{r}}_2^{(1)}\right) = \hat{\boldsymbol{y}}^{(2)}$$

- Or, in matrix form:

$$\hat{\boldsymbol{r}}^{(2)} = R^{(1)}\boldsymbol{w}^{(2)}, \text{ where } \boldsymbol{w}^{(2)T} = \begin{bmatrix} w_0^{(2)} & w_1^{(2)} & w_2^{(2)} \end{bmatrix}.$$

The logistic function

- The difference between linear and nonlinear regression is the neurons.
- Each neuron applies a linear or nonlinear function to the weighted inputs.
- The most common nonlinear function is the logistic (sigmoidal) function, shown here.
- The mathematical form of the logistic function is:



$$f(y) = \frac{1}{1 + \exp(-y)}$$

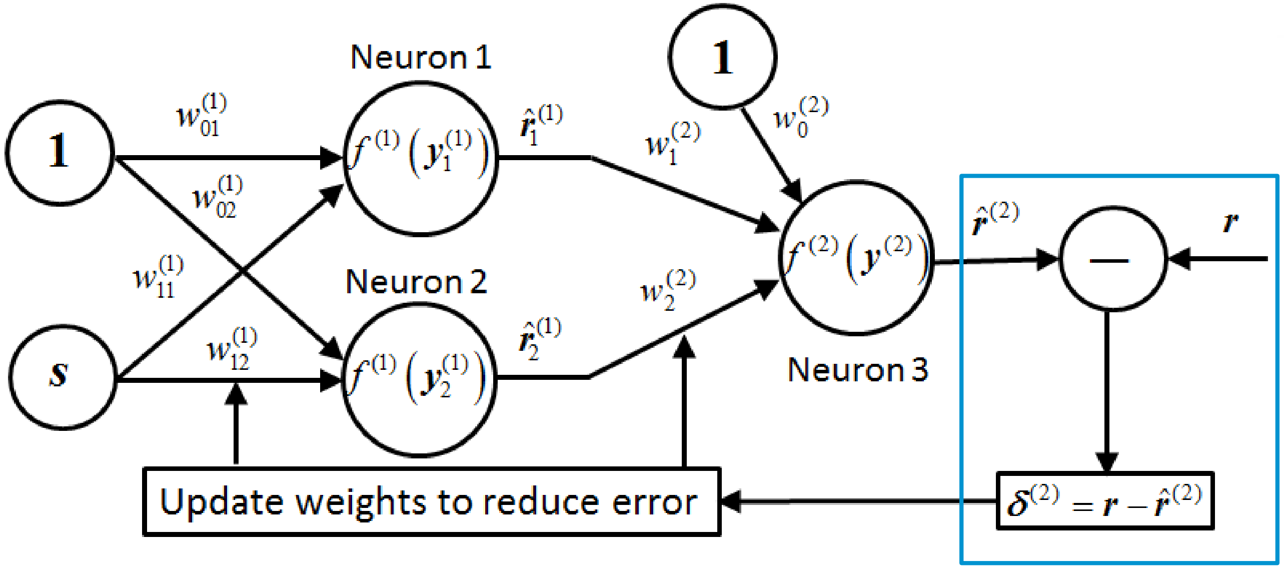
- A computational advantage of the logistic function is its derivative, given by:

$$f'(y) = f(y)\left(1 - f(y)\right)$$

The backpropagation algorithm

- To find the optimum weights for our neural network a procedure called error backpropagation is used.
- The algorithm can be summarized as follows:
  - Initialize the weights in both layers to small random values.
  - Starting with the weights in the output layer, change the weights so as to minimize the error between the computed and desired output values.
  - Backpropagate the error minimization for all layers.
  - Iterate until an acceptable error is found.
- Let’s now look at error backpropagation in more detail.

The backpropagation algorithm



- The first step is to compute the error between the output of the network (starting with the initial weights), which is given by:

$$\delta^{(2)} = r - \hat{r}^{(2)}$$

The backpropagation algorithm

- The details of the backpropagation algorithm are as follows.
- We start by iteratively updating the weights in layer 2, where for the k+1st iteration, we get:

$$\boldsymbol{w}_{(k+1)}^{(2)} = \boldsymbol{w}_{(k)}^{(2)} + \alpha R_{(k)}^{(1)T} \boldsymbol{\delta}_{(k)}^{(2)}$$

- With the exception of the reflectivity matrix note the similarity of this equation to the steepest descent equation.
- The second step is to update the weights in layer 1 as follows:

$$W_{(k+1)}^{(1)T} = W_{(k)}^{(1)T} + \alpha S \boldsymbol{\delta}_{(k)}^{(1)}, \text{ where:}$$

Derivative of logistic function

$$\boldsymbol{\delta}_{(k)}^{(1)} = \boxed{\left[R_{(k)}^{(1)} \circ (1 - R_{(k)}^{(1)}) \right]^T} \circ \left[\boldsymbol{w}_{(k+1)}^{(2)} \boldsymbol{\delta}_{(k+1)}^{(2)T} \right]$$

- The key new idea in backpropagation is the use of the derivative of the logistic function, where o implies an element-by-element multiplication of two matrices.