Project 3: Shale Caprock Brittleness Estimation for CO2 Storage Potential using Neural Networks and Random Forest Regression

Owen Huff, Romain Corseri, and Ilya Berezin

December 16th, 2022

FYS-STK 4155 - Applied Data Analysis and Machine Learning

University of Oslo

Abstract

Shale rock layers are commonly employed as caprocks for subsurface CO2 sequestration. The brittleness of these shale rocks is an important property to estimate, because highly brittle shales can fracture and allow CO2 to escape from storage reservoirs. To this end, we develop machine learning models that can predict shale brittleness from 1-dimensional seismic data. We first build a mineralogical model for shales that can be used to generate densities and seismic velocities for a given shale clay content. Then we generate synthetic seismograms by convolving the product of these densities and velocities with a source wavelet extracted from seismic data in the North Sea; these seismograms act as our training data. We also convert the clay content value for each synthetic to a brittleness value using an empirical mapping equation; these brittleness values act as our training labels. Three different models are trained: a random forest regressor, a feed forward neural network, and a convolutional neural network, and all reach similarly low values of error on the testing dataset, approximately 1.2e-4 for the mean squared error and 0.97 for the R2 value. We then use the trained CNN model to predict on real seismic data collected in the North Sea that images the Draupne Shale Formation. Predictions applied to traces from this dataset yield increasing shale brittleness values from 0.33 to 0.37 as offset increases along the seismic section, indicating possible lateral heterogeneity in brittleness. The brittleness values predicted on this real data are approximately the same as what earlier studies on the same shale yielded using a more complex mineralogical model and elastic properties methods.

Reproducibility

The code used for this report can be accessed at:
https://github.com/orhuff/FYS-STK-4155-Project3-Shale-Brittleness-Network

Introduction

Carbon sequestration is a promising method for reducing greenhouse gas emissions. It works by injecting gaseous CO2 up to thousands of meters deep in the subsurface within sedimentary rock formations (Benson and Cole, 2008). The rock formations that are typically good for storing CO2 are sandstones, as they have a large amount of pore space (high porosity) and permeability (ease of flow) to hold and receive fluids. These storage rocks are called reservoirs, and are similar in concept to petroleum or water reservoirs. However, because of its low density, CO2 can escape and migrate to undesired locations if there are not appropriate sealing formations above the reservoir rocks. The rock type that commonly acts as a seal is shale, which is a dark colored sedimentary rock with low permeability. This low permeability ideally prevents CO2 (or oil and gas in the case of hydrocarbon extraction) from escaping the reservoir. Thus, shale formations lying above a CO2 sequestration reservoir are typically referred to as "caprocks."

But not all shales act as good caprocks. This is because some compositions of shale have higher brittleness, meaning they can fracture more easily. When fractures form in shale caprocks, sequestered CO2 can escape from below. Some shales are more brittle, such as the Eagle Ford Formation in southeastern Texas, which is used for hydraulic fracturing operations to extract oil and gas. Other shales are less brittle (or more ductile), such as the Draupne Formation in the North Sea, which has been identified as a promising caprock for CO2 sequestration (Gabrielsen et al., 2021). However the brittleness can also vary in space along the same shale layer, because of heterogeneities in mineral composition and how the rock was deposited in different locations. These factors motivate the need for methods that can estimate the brittleness of shale rocks deep in the subsurface, since it can help in identifying good (and poor) quality sealing locations for storing CO2 below.

Some current methods to estimate shale brittleness include experiments where powerful machines apply compressive or shear stress to shale rock samples until failure to measure their strength (Andreev, 1995; Tarasov et al., 2013). And there also exist more precise methods like microindentation, where very small and precise forces are placed on the surface of a rock sample in order to measure its response (Yang et al., 2022). However, it can be difficult and time consuming to collect these samples of shale, and they only are obtainable at drilled well locations, which can be far apart in space. Therefore there has been a push towards methods for estimating brittleness that are more generalizable. These include building mineraological models of shale that can be used to predict brittleness, and using the elastic properties obtained from well and seismic data (Rahman et al., 2022).

Machine learning methods are an attractive avenue for solving problems in geology and geophysics. There are examples of models being trained to extract faults (sliding discontinuities in rock formations) and horizons (interfaces between rock formations) from seismic data using convolutional neural networks (Wu et al., 2019; Yang and Sun, 2020). Furthermore, random forest models have been used to identify and classify different rock layers in seismic data (Kim et al., 2018), and feed forward neural networks have been used to detect seismic signals and differentiate them from noise (Wu and Lin et al., 2019). Therefore

all of these types of models are promising for our task of learning to predict shale brittleness from seismic data.

In this project we estimate shale brittleness via machine learning models trained on synthetic seismic data. A mineralogical model of clay content for shales in our study area is first developed and used to estimate seismic velocities and densities. These velocities and densities are convolved with a seismic source wavelet in order to generate realistic synthetic seismograms for different clay content values, which act as our training data. Concordantly, we use an empirical equation to convert clay content values to shale brittleness, which act as our training labels. A random forest, feed forward neural network, and convolutional neural network are then trained to predict brittleness values from the synthetic data. Finally, we test a trained network on real seismic data from the Norwegian North Sea to estimate the brittleness of a shale layer.

Theory

Shale Mineralogy and Brittleness

We first describe a mineralogical model of shale that can be related to brittleness, and used to generate realistic seismic data. Shale is usually predominantly composed of clay minerals, quartz, and carbonates. However in our study area, the Norwegian North Sea, the concentrations of carbonates tends to be low. Therefore we build a simplified mineralogical model of shale using clay minerals, quartz, and the fluid-saturated pore space of the rock. The two dominant types of clay minerals in our study area are kaolinite and smectite, and microscopic analysis of shale samples taken from wells has shown their relative occurrence (as a fraction of the total amount of clay) to be about 70% kaolinite and 30% smectite (Gabrielsen, 2021). The relative occurrence of total clay minerals and quartz can vary more significantly, however, ranging from about 20% clay / 80% quartz on one end to 80% clay / 20% quartz on the other (Rahman et al., 2022). Therefore to construct our mineralogical model we decide to use a normal distribution of clay fraction, with a mean of 50% and a standard deviation of 8%. For the porosity, which is the volume fraction of the rock that is empty (or fluid-saturated) space, we use a value of 25%, which is informed by well log measurements in our study area.

Each of these minerals (kaolinite, smectite, and quartz) has a typical density value, and a speed at which seismic compressional waves (P-waves) pass through it. We can use these values to construct a model for how seismic waves will travel through the shale. The values of density and seismic P-wave velocity for these materials are presented in Table 1. Note that the fluid (usually mostly water) contained within the pore space of the shale also has a density and seismic velocity that will affect the modeling.

Mineral Seismic Properties	Kaolinite	Smectite	Quartz	Water
Density (g/cm^3)	2.616	2.613	2.65	1.0
P-wave Velocity (m/s)	3020	5060	6040	1500

Table 1: Density and P-wave seismic velocities for minerals (and water) used in the shale mineralogical model.

With these mineral properties and their relative occurrence in shale as described above, we can formulate empirical equations for the density ρ and seismic velocity Vp of a shale caprock layer as follows:

$$\rho_{shale} = (1-\phi) * \left[(k_f * \rho_k + s_f * \rho_s) * c_f + \rho_q * (1-c_f)\right] + \phi * \rho_w \tag{1}$$

$$Vp_{shale} = (1-\phi) * \left[(k_f * V_{pk} + s_f * V_{ps}) * c_f + V_{pq} * (1-c_f)\right] + \phi * V_{pw} \tag{2}$$

In these equations, ϕ is the shale porosity, c_f is the fraction of the total rock mineral content that is clay, k_f is the fraction of clay that is kaolinite, s_f is the fraction of clay that is smectite, ρ_k is the density of kaolinite, ρ_s is the density of smectite, ρ_w is the density of water, V_{pk} is the p-wave velocity of kaolinite, V_{ps} is the p-wave velocity of smectite, and V_{pw} is the p-wave velocity of water. Most of these quantities are again listed in Table 1, while c_f is estimated empirically as a normal distribution $\sigma(0.5, 0.08)$, and ϕ is estimated from well logs as 0.25.

There is actually not an objective definition of a mineralâs brittleness, as it is dependent on too many factors to be described by an analytical equation. Some of these factors include the mineralogy, stress, texture, temperature, pressure, and others (Rahman et al., 2022). However there are a few methods which have yielded strong results. Two of these include the elastic properties method and the mineral composition method. The elastic properties method uses Youngâs Modulus E_s, which is the ratio of stress to strain that a material experiences, and Poissonâs ratio ν, which is the ratio of the amount of transverse stress to longitudinal stress that a material experiences, in an empirical equation as such to calculate the brittleness B (Grieser and Bray, 2007):

$$B = \frac{1}{2}\frac{E_s E_{min}}{E_{max} - E_{min}} + \frac{\nu - \nu_{max}}{\nu_{min} - \nu_{max}} \tag{3}$$

Where E_{max} is 69 GPa, E_{min} is 0 GPa, ν_{max} is 0.5, and ν_{min} is 0. The Youngâs moduli and Poisson ratios can be calculated from P-wave and S-wave (shear wave) velocities as described in Rahman et al., 2022. This data is available often available in wells due to the collection of sonic logs.

However, wells are often separated by large distance and there can be significant spatial heterogeneities between the wells. Therefore other methods which are more generalizable are more desirable, such as the mineralogical method, which we apply in this project. This method assumes that the brittleness is equal to the fraction of brittle minerals compared to the total mineral content (Rahman et al., 2022). While some minerals are definitely brittle (quartz) and others definitely ductile (clay) there are others that are somewhat brittle and usually make up a smaller fraction of the mineral content (especially in the North Sea), such as carbonates, feldspars, pyrite. Rahman et al. use a slightly more sophisticated mineralogical method than our assumption of only clay and quartz, and also include more types of clay minerals such as illite. Therefore we decide to map the values from our simpler model to their observed brittleness indices, which are thought to be more accurate.

We use measurements of clay fraction and shale brittleness index presented in Rahman et al., 2022 to formulate an empirical mapping between these two quantities. While clay fraction is approximated by the normal distribution $\sigma(0.5, 0.08)$ which essentially spans the range of 0.2 to 0.8, brittleness in our study area only spans the range of 0.2 to 0.6, with extreme values being very uncommon (Rahman et al., 2022). Therefore to obtain brittleness from the clay fraction, we apply a linear mapping transformation to the clay fraction values as follows:

$$B = \frac{c_f - c_{fmin}}{c_{fmax} - c_{fmin}} * (B_{max} - B_{min}) + B_{min} = \frac{c_f - 0.2}{0.8 - 0.2} * (0.6 - 0.2) + 0.2 \tag{4}$$

These brittleness values will act as the labels - or what we want to predict with our machine learning models - corresponding to the seismic response of a shale caprock of a given mineral composition.

Translation to Seismic Data

We next describe how the mineralogical and seismic properties of our shale caprock (and the rocks above and below it) can be translated into seismic data, which will serve as the training data for our machine learning models. Rock layers follow a depth profile in the subsurface, and have velocity and density values as described earlier. The product of the P-wave velocity and the density along the depth profile yields a quantity called the acoustic impedance (Aki and Richards, 2002). Acoustic impedance can be thought of as a resistance to (or pressure against) the propagation of a seismic wave, similar to electrical impedance within circuits.

$$I = V_p * \rho \tag{5}$$

Every time a seismic wave encounters an interface (a new rock layer) with a change in impedance, some of the energy is transmitted and some is reflected. The fraction of energy that is reflected at normal incidence is defined by the reflection coefficient, which is calculated

from the impedances of the layers above and below the interface as such (Aki and Richards, 2002):

$$R = \frac{I_{lower} - I_{upper}}{I_{lower} + I_{upper}} \tag{6}$$

A series of reflection coefficients can then be convolved with a wavelet in order to obtain a synthetic seismogram. The wavelet is a representation of the seismic source, and the amplitudes and frequencies it contains. In the case of marine seismic data the source wavelet is a pulse from an airgun that creates pressurized explosions just below the water surface, though in land seismic data it can be from dynamite, a vibrating truck, or other devices. Convolution is a sliding integration operation that allows the seismic wavelet to impart its frequency and amplitude information on the reflectivity series. It is often denoted by the $*$ symbol as such, with the convolution between two functions f and g:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau \tag{7}$$

This workflow, starting from a log of lithologies and their associated p-wave and density values, and generating a synthetic seismogram of amplitudes, is shown pictorially in Figure 1. We will follow this process in order to generate synthetic seismic training data from velocity and density values that are informed by our shale mineralogical model.

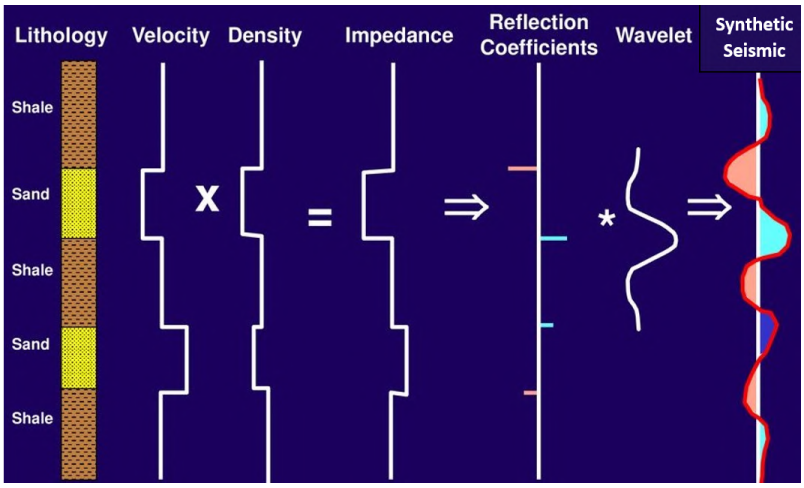

Figure 1: Diagram of the workflow for generating a synthetic seismic trace from velocity and density values with respect to depth (Shroeder, 2006).

Models

Random Forest Regressor

Random forest models are a collection of many decision trees that can be trained for regression or classification tasks. In this project we focus on the regression form since our goal is to predict a continuous brittleness value. Decision trees are a method to learn from data based on a series of hierarchical if-else conditions (Mehta et al., 2019). Each decision tree has a root node, where input data is fed into. Then the data is divided according to decision nodes (or branches), which impose conditions or tests on the data. This process continues up the tree as decision nodes can be split into multiple nodes. When there aren't any further decisions or conditions imposed, the final nodes are called terminal nodes (or leaf nodes, in keeping with the tree metaphor). In this way decision trees can be trained to learn complex patterns in the data.

However, decision trees by themselves have limitations, such as a proclivity towards overfitting to the training data. This is where random forests come in. A random forest is a collection of decision trees, where the individual trees are trained on subsets of the data. Then in order to make a final prediction, the predicted values from the individual trees are averaged. This leads to more generalizable models and less chance of overfitting compared to the individual decision tree approach (Mehta et al., 2019).

Random forests have a significant number of hyperparameters which affect their performance. The most straightforward parameter is the number of estimators, which is the number of decision trees within the random forest. Essentially, the higher this number is the better, though there is very often a point of diminishing returns reached quite quickly. The maximum depth is the depth of the decision trees, which determines the number of splittings that the tree is allowed to make. Maximum features are the maximum number of features allowed to be used when training the trees. The final two important hyperparameters are minimum samples at a leaf, which sets a minimum of how many data samples are allowed following all of the splittings, and minimum samples to split, which sets a minimum number of samples that are necessary to split a decision node. These hyperparameters can be optimized using a cross validation grid search to see which result in the best-performing model; we implement in our random forest regressor case.

Feed Forward Neural Network

Artificial neural networks are nonlinear statistical models that can learn to approximate any complex function to a desired uncertainty. They are designed to mimic a human brain, aggregating neurons within a layer that send signals to other neurons in a different layer. Each neuron accumulates its incoming signals which must exceed a given threshold to yield an output, as defined by an activation function. Each connection is represented by a weight variable and a bias term. Neural networks are called feed-forward when the information flows just in one direction through the model, from the input through all the layers, and producing an output (Nielsen, 2015).

In Figure 2, a network diagram of a multilayer perceptron (MLP) or feed forward neural network (FFNN) is depicted. This is a fully connected network with a single input layer containing four neurons (x_p), one hidden layer with 5 neurons and an output layer with 2 neurons (y_k). Each circle represents neurons (or nodes), and the weight parameters are represented by links

between the neurons. The number of hidden layers define the depth of the network. Each node of the hidden layer is associated with an activation function taking the weighted sum and bias as input (z_h). Some of the common activation functions, a few of which we will implement in our models, are described next.

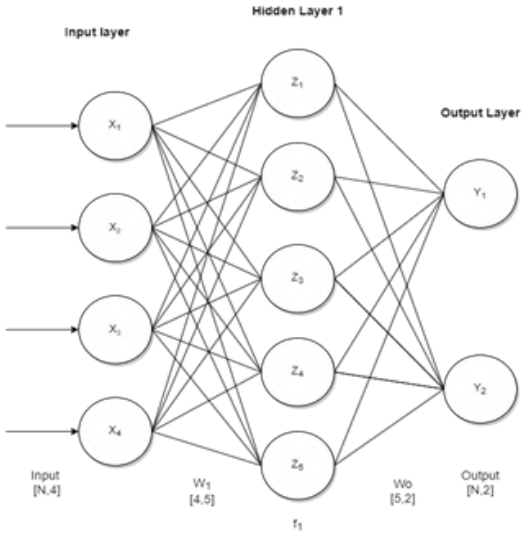

Figure 2: Diagram of a feed forward neural network.

The choice of activation layers can have a influence on the performance of the neural network. In a FFNN the activation functions are required to be non-constant, bounded, monotonically increasing and continuous. Common choices include the sigmoid function and the Rectified Linear Unit function (ReLU), both of which we implement in this report. A description of each activation function is provided below.

The sigmoid (or logistic) activation function is designed so as to take an input value, and yield an output between 0 and 1. It is defined by the following equation:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{8}$$

The next activation function we will implement is the ReLU, or Rectified Linear Unit. The ReLU is a piecewise function as defined below. It is less computationally expensive than the sigmoid function, and does not easily experience the so-called vanishing gradients problem (where the output gradients are very close to zero) because its derivative is a non-zero constant.

$$\sigma(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases} \tag{9}$$

A feed-forward neural network (FFNN) operates as follows: each input x_p is multiplied by a

weight w_i, that indicates the strength of the connection to each node in the next layer. Then a bias is added to each of the weighted sums and for each neuron. The biases and weights are usually initialized with random small numbers distributed around zero. The result is passed through an activation function $\sigma(z)$, providing the output for each node, which in turn becomes the input for the next layer. The process is repeated for an arbitrary number of hidden layers and their respective nodes until the output layer is reached. Finally, the output $\tilde{y}$ is compared to the target response y, using an loss function of choice such as the mean squared error for regression tasks.

After one feed-forward pass through the network (i.e., one iteration), the weights and biases are adjusted to minimize the errors in the output. This task is performed by the back propagation algorithm (Nielsen, 2015). The basics of the algorithm are described below. After a feed-forward pass, the output error (last layer L) is given by:

$$\delta_j^L = f'(z_j^L)\frac{\partial C}{\partial(a_j^L)} \tag{10}$$

where $f'(z_j^L)$ is the derivative of the activation function of the last j neurons, a_j^L is the output data of the last j neurons, and C is the cost (or loss) function. The back-propagated error can be computed for each layer l = L-1, L-2, ... , 2 as (Nielsen, 2015):

$$\delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1} f'(z_j^l) \tag{11}$$

Weights and biases can now be updated using gradient descent for each layer l = L-1, L-2, ... , 2 according to:

$$\begin{aligned} w_{jk}^{l+1} &= w_{jk}^l - \eta \delta_j^l a_k^{l-1} \\ b_j^{l+1} &= b_j^l - \eta \frac{\partial C}{\partial b_j^l} = b_j^l - \eta \delta_j^l \end{aligned} \tag{12}$$

Where η is the learning rate. This learning rate can be tuned with a variety of algorithms such as RMSProp, AdaGrad, and Adam. The advantage of tuning the learning rate is that it can result in faster convergence to a more accurate minima of the loss function. For our neural network model training we implement Adam, whose name derives from adaptive moment estimation. The key concept of the Adam algorithm is that it keeps track of a running average of the squared gradients, as well as a running average of the unsquared gradients (Kingma and Ba, 2014). The update rule for Adam is:

$$x_{k+1} = x_k - \frac{\eta}{\sqrt{\hat{v}_k} + \delta}\hat{m}_k \tag{13}$$

Here the biased running average of the gradients is referred to as the first moment $\hat{m}$, and the biased running average of the squared gradients is referred to as the second moment $\hat{v}$. We refer the reader to the Theory section of our Project 2 report for more details on feed forward neural networks and learning rate tuning methods.

Convolutional Neural Network

One-dimensional convolutional neural networks (1D CNNs) are a type of deep learning model that use convolutions over 1-dimensional input data such as text or time series, allowing for efficient extraction of features from long sequences.

A 1D CNN is a type of neural network which operates on one-dimensional data, such as time series and audio signals. In contrast to convolutional neural networks that operate on two-dimensional data, like images, 1D CNNs allow for the extraction of temporal patterns from sequences by applying linear convolutions to them (Kiranyaz et al., 2021). These are actually the same convolution operations that we described in the earlier section on translation to seismic data.

A basic 1D CNN consists of several convolution layers and max-pooling layers arranged in succession. In each convolution layer, an array of filters is applied to the input sequence. The output of the convolution layer is given by the following equation:

$$y_i^{(l)} = f\left(\sum_{j=0}^{N-1} a_j^{(l)} x_{i+j} + b^{(l)}\right) \tag{14}$$

Where x is the input sequence, $a^{(l)}$ and $b^{(l)}$ are filter coefficients, and f represents an activation function. Each filter extracts a certain feature from the input sequence, and together they form a feature map which can be used for classification or regression tasks. After each convolution layer, a max-pooling layer is used to reduce dimensionality as well as combine features from various parts of the sequence. The output of the pooling layer can be represented as:

$$z_i^{(m)} = \max_{j \in S_i} y_j^{(m)} \tag{15}$$

Where S_i denotes the set of neurons fed into the pooling unit z_i, and $y_j^{(m)}$ is the output vector of neuron j. Max pooling layers are followed by dense layers which transform the pooled outputs into output vectors suitable for classification or regression tasks.

A typical 1D CNN is composed of an input layer, one or more convolutional layers, an activation function, a pooling layer and an output layer. The input layer takes in a sequence of numbers, which can be thought of as representing a signal over time. This signal is then processed by convolutional layers that are comprised of sets of kernels that slide across the input sequence and detect patterns in the signal (Kiranyaz et al., 2021). The activation function serves to non-linearly transform the signals coming out of the convolutional layers so they can be used in

subsequent layers; commonly used activation functions include ReLU and sigmoid functions. Following this, a pooling layer is applied to reduce dimensionality and introduce invariance to small changes in the signal; usually max pooling or mean pooling is used (Goodfellow et al., 2016). Finally, after successive convolution/activation/pooling steps have been performed on each channel, the output from each channel is combined into a single vector which constitutes the output from the network. An example diagram of a 1D CNN is shown in Figure 3 below.

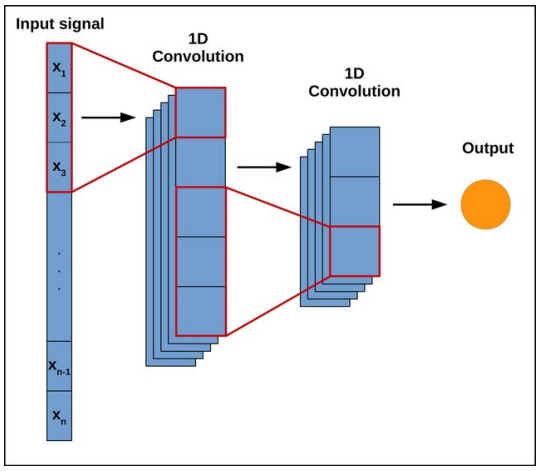

Figure 3: Diagram of a 1D CNN. (Shenfield and Howarth, 2020)

The mathematics behind 1D CNNs can be broken down into two parts: calculating filter weights and applying those filters to an input signal. To calculate filter weights, we first need to define our kernel size – that is, how many elements each filter operates over – as well as our stride size – how far apart each element in our kernel should be placed when sliding across our sequence (Kiranyaz et al., 2021). Denoting these values respectively with K and S, every position in our sequence will then have its own set of $K+1$ filter coefficients: $w_0, \ldots, w_K$ where w_i represents how much weight should be given to element i within our kernel at any particular position within our sequence. These filter coefficients are determined through training using stochastic gradient descent (SGD).

Applying these filters to an input signal involves sliding our kernel across all possible positions within it using a stride size S; for example, if we had an input vector $X = [x_1, \ldots, x_n]$ with n elements and wanted to apply a 5 element kernel with stride $S = 2$, then we could slide it across X starting at x_1 till x_{n-4}. At each position p, we would take the dot product between $W = (w_0, \ldots, w_K)$ and $X = (x_p, \ldots, x_{p+K})$ such that $W \cdot X = \sum_{i=0}^{K}(w_i \cdot x_{p+i})$; this value would become one element in a resulting vector $Y = [y_1, \ldots, y_{n-4}]$ with $n-4$ elements where $y_i \equiv W \cdot X$ at position $i+1$ within X for $0 < i < n-4$. This process can be expressed algebraically using the convolution operation: $Y = X \otimes W$ where $\otimes$ denotes the discrete circular convolution operator typically defined in terms of infinite sums such as $\sum_{j=-\infty}^{\infty} X(j) \cdot W(\text{mod}(i-j))$ for $0 \leq i < n-4$ (Kiranyaz et al., 2021).

In summary, 1D CNNs are powerful models capable of extracting complex temporal patterns from long sequences through efficiently sliding multiple kernels along them; they have been

shown to excel at tasks such as sentiment analysis or speech recognition due their ability to identify important trends while ignoring minor fluctuations in their inputs. Through careful design of their kernelsâ sizes and strides they can also allow us control over how much information needs to go into later stages before producing outputs; this has enabled them to achieve state-of-the-art results on many challenging problems where there may exist noise or redundancy inherent within their inputs. Finally by taking advantage of existing implementations like Tensorflow we can easily train 1D CNNs on large datasets without having needing write complex algorithms ourselves reducing both development time cost while still gaining access impressive accuracy gains on various tasks involving temporally varying inputs.

The use of 1D CNNs has been demonstrated in many machine learning tasks including speech recognition, object tracking, motion detection, classification and anomaly detection (Kiranyaz et al., 2021). Furthermore, 1D CNNs have been shown to outperform traditional methods such as Recursive Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks due to their ability to learn temporal dependencies more effectively than those methods can. Therefore we see it as an appropriate method for processing and analyzing 1D seismic signals.

Method

Synthetic Training Data Generation

We follow the workflow shown in Figure 1 in generating our synthetic training data. First, depth profiles of density and p-wave velocity are constructed to be 101 samples long, with 1 sample being equal to 10 meters in depth. We set the shale caprock to be 10 samples, or 100 meters thick, which is a typical thickness of the Draupne shale in our study area (Rahman et al., 2022). Above the caprock (the overburden), we assume a sandstone medium with velocity 2400 m/s and density $2.24\ g/cm^3$, and below the caprock (the reservoir or underburden), we also use a sandstone medium but with velocity 3000 m/s and density also $2.24\ g/cm^3$. These density and velocity values for the overburden and underburden are estimated from sonic log measurements of seismic velocity in a nearby well (Fawad et al., 2021). The density and velocity of the shale caprock are then calculated using the empirical equations (1) and (2), respectively. From these profiles of density and velocity for the 3-layer model, the workflow in Figure 1 is followed in order to generate a given synthetic seismogram. In the final step of the synthetic generation, convolution of the reflectivity series with the wavelet, we use a wavelet that was statistically extracted from seismic data in our study area. This wavelet is shown below in Figure 4.

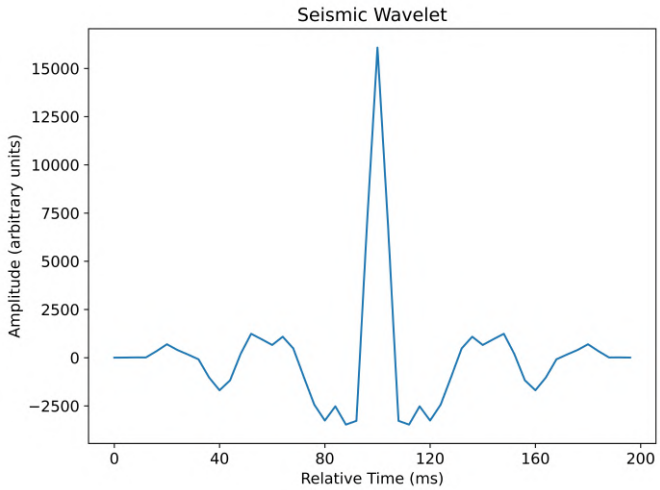

Figure 4: Seismic source wavelet used in our synthetic generation. It was statistically extracted from seismic data in our study area at the Draupne Formation depth.

Each synthetic is generated for a given clay fraction, which again is given by a normal distribution $\sigma(0.5, 0.08)$. We sample from this normal distribution 3000 times, and thus create 3000 synthetic seismograms. The clay fraction values for each synthetic seismogram are then converted to brittleness index values according to equation (4). These brittleness values act as the labels that we want to predict for each training synthetic. In Figure 5 we present an example of the data used in each step of the workflow, with a final synthetic seismogram result. The large positive amplitude spike at 500 meters represents the top of the caprock, and the relatively smaller negative amplitude spike at 600 meters represents the bottom of the caprock.

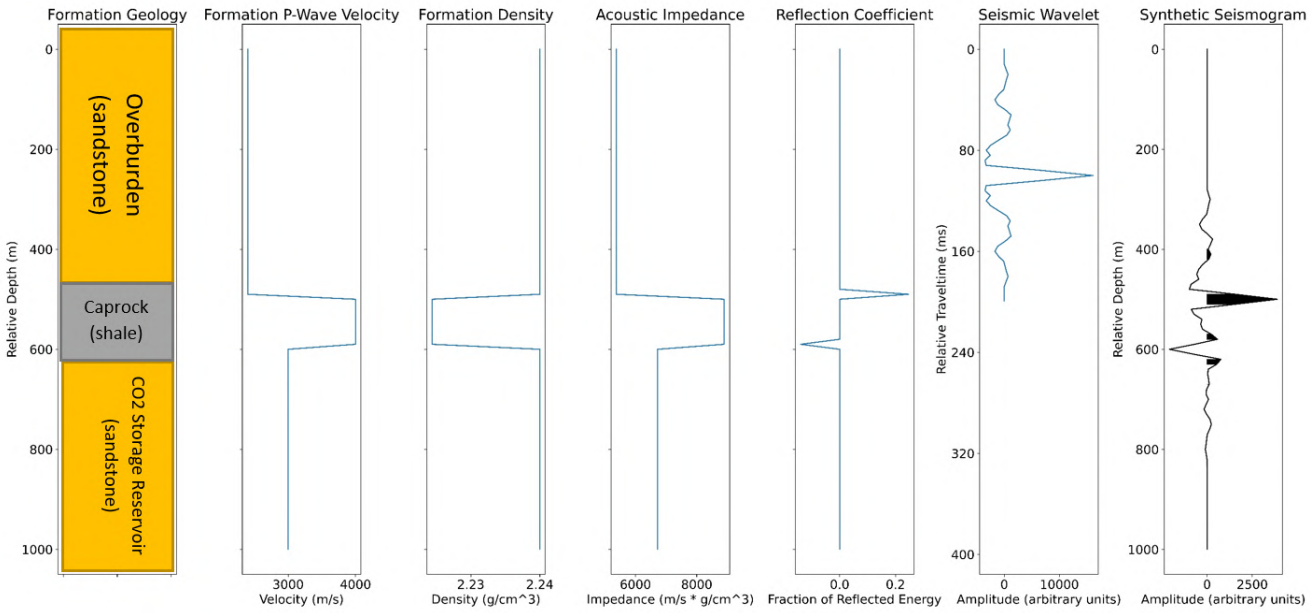

Figure 5: The real implementation of the workflow presented earlier in Figure 1, with the end result being a synthetic seismogram.

To simplify our generated synthetic seismograms, we zero out the amplitudes more than 100 meters away from the top and bottom of the caprock. This enables us to focus only on the caprock region and achieve better results when we later predict on real, noisy field data. An example synthetic before and after the zeroing out is shown in Figure 6.

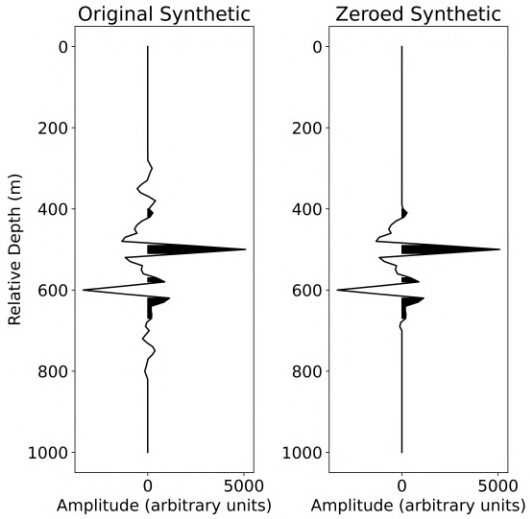

Figure 6: Left - synthetic seismogram before zeroing. Right - synthetic seismogram with amplitudes zeroed out that are more than 100 meters away from the caprock.

The brittleness values of the generated synthetics approximately follow a normal distribution with mean 0.4 and standard deviation 0.07 (because they were mapped from the clay fraction

normal distribution). The distribution of clay fraction and brittleness for the generated synthetics is shown in Figure 7.

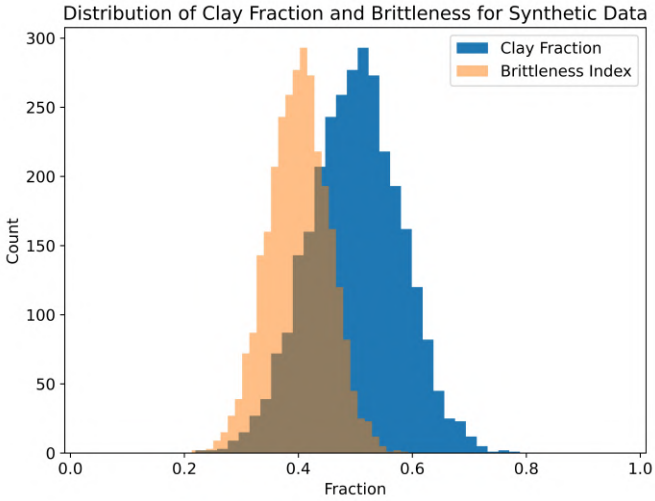

Figure 7: Distribution of clay fraction and brittleness values for the 3000 generated synthetics.

The generated synthetic with the highest brittleness (0.593) and the lowest brittleness (0.213) are shown in Figure 8. Note that the amplitudes for the lowest brittleness synthetic are significantly higher than for the highest brittleness synthetic. This is because increasing the clay content results in significantly increasing the amount of kaolinite, whose high velocity causes increased velocity (and therefore impedance) contrast with the overburden rock.

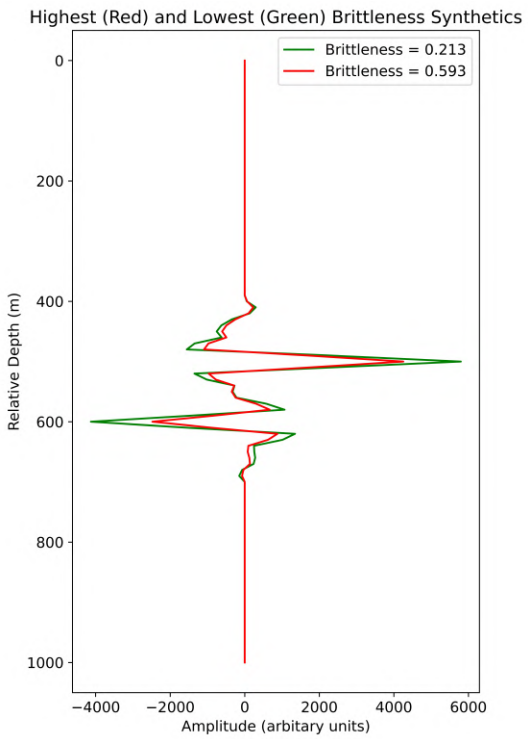

Figure 8: The lowest (green) and highest (red) brittleness synthetics generated.

Using the 3000 synthetics, we split them into a train set of 70% (or 2100) and a test set of 30% (or 900). We also normalize the training and testing synthetics by the maximum amplitude in the data, using the MinMaxScaler in scikit learn. However, we do not normalize the brittleness label values because they are already between 0.2 and 0.6, and we want to retain this information.

Random Forest Regressor Parameterization and Training

The first type of model that we train on our synthetic seismic data is a random forest regressor. As mentioned in the theory section, random forests have many tunable hyperparameters. Thus we perform a grid search with cross validation over these hyperparameters in order to find the ones that perform the best. This is simple to do using scikit learn's function GridSearchCV, which enables you to input a grid of hyperparameters, a cross validation fold (we use 3), and the estimator (or model) in question. The hyperparameters we search over are presented in Table 2, and the combination of those that yield the lowest training error are highlighted in green.

Random Forest Hyperparameters	Best parameter in green			
max_depth	2	4	6	8
max_features	2	3		
min_samples_leaf	3	4	5	
min_samples_split	8	10	12	
n_estimators	100	200	300	1000

Table 2: Results of the hyperparameter grid search with 3-fold cross validation, for the random forest regressor model. The best hyperparameters are chosen as the ones the resulted in the lowest error in the model training.

Feed Forward Neural Network Architecture and Training

Next we construct a feed forward neural network in Tensorflow to train and predict on our seismic data. We use a simple architecture, with an input layer of 101 (the length of a given synthetic seismogram), a hidden layer of 250 nodes with ReLU activation, followed by a flattening layer and output layer with 1 node (because we want to predict 1 brittleness value).

For the output activation, we decide to use a scaled version of the sigmoid function. This is because we want to have output brittleness values be between 0.2 and 0.6, as opposed to the bounds of 0 to 1 of the regular sigmoid function. We define this scaled sigmoid function as such:

$$\sigma(z)_{scaled} = \frac{1}{1+e^{-z}} * (B_{max} - B_{min}) + B_{min} \tag{16}$$

where B_{max} and B_{min} are the maximum and minimum brittleness values of 0.6 and 0.2, respectively. We implement our feed forward neural network in Tensorflow, and load this scaled sigmoid function as a custom object. In training the network, we use the mean squared error as the loss function, use the Adam optimizer with an intial learning rate of 1e-5, use a minibatch size of 25 data samples, and train for 100 epochs. Each epoch takes approximately 1 second to complete. The training loss curve is shown in Figure 9, where the training error converges to about 1e-5.

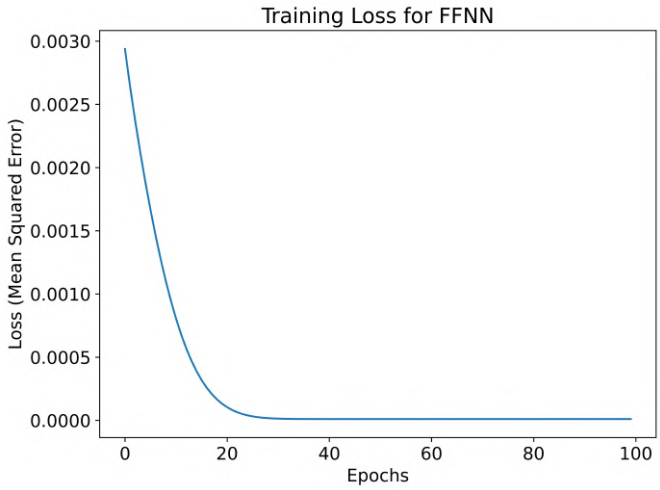

Figure 9: Training loss curve for the feed forward neural network.

Convolutional Neural Network Architecture and Training

Our final type of predictive model is the 1D convolutional neural network (CNN). We use the architecture shown in Figure 10 below. The dimensions of the input signal (101 samples by 1, which is necessary for Tensorflow input) are shown in the green box. The signal then passes through two convolutional layers with 16 filters each. These convolutions reduce the length of the signal by 2 each time, and increase the dimensionality to the number of filters. This is followed by a maxpooling layer, which downsamples the signal by a factor of 2 in order to avoid overfitting and identify the most important features. Then two more convolutional layers are applied, this time with filter numbers of 32. Following another maxpooling layer, the convolutional features are flattened and passed through a dense layer of 250 nodes with ReLU activation. Finally, the output is fed through a single node dense layer with our scaled sigmoid activation function, yielding a predicted brittleness value.

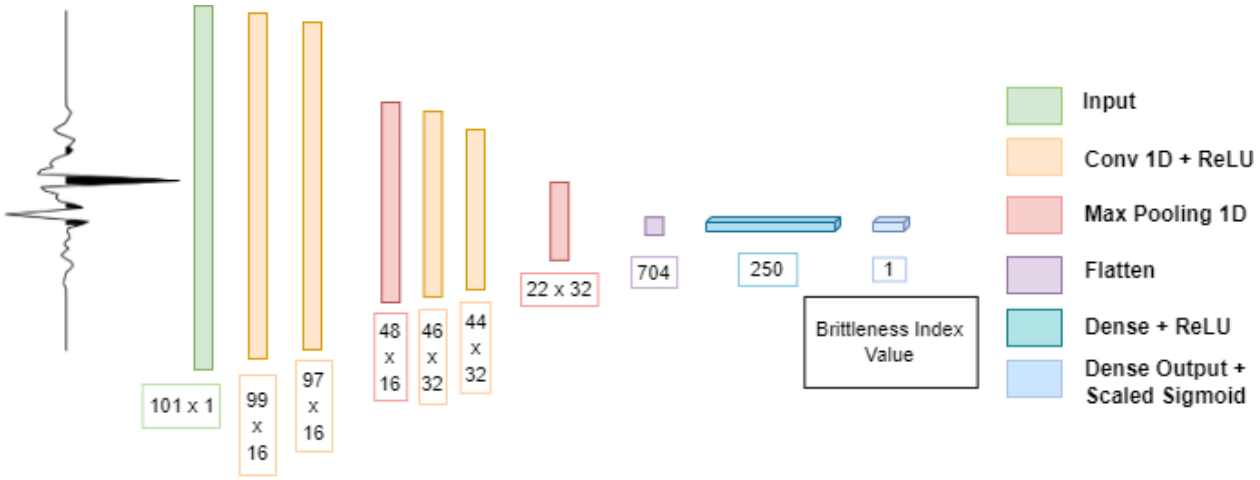

Figure 10: 1D CNN network architecture.

We use the same training parameters as for the FFNN, and show the results of the training loss in Figure 11. This training loss converges to a value of about 1.54e-6.

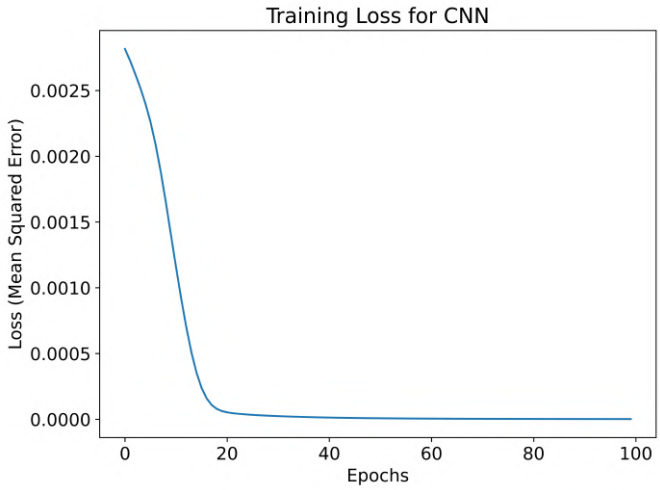

Figure 11: Training loss curve for the 1D convolutional neural network.

Synthetic Results

Using the best-performing hyperparameters from the GridSearchCV, we construct a random forest regressor model and fit it to our normalized training data. Then we predict brittleness values on the holdout testing data, and calculate the error between the predictions and the actual values. For the mean squared error, we find a value of 1.23e-4, and for the R2 value, we find a value of 0.967. The high R2 value indicates that the random forest model is yielding accurate predictions on the testing data. We repeat this process with the FFNN and CNN whose training was shown in the previous section. The testing error and R2 results for all three models are shown in Table 3 below. The FFNN may achieve slightly better results than the random forest and the CNN, however the results are practically equivalent.

Model	Test MSE	Test R2
Random Forest	1.23e-4	0.967
FFNN	1.17e-4	0.968
CNN	1.23e-4	0.967

Table 3: Testing error and R2 results for the different trained models.

We choose to present further results using only the CNN model. This is for a few reasons: 1) The CNN model achieved the lowest training error of the three model types, 2) the testing errors returned by all three models are practically equivalent so this enables more brevity in the presentation, and most importantly, 3) we plan in the future to increase the complexity of

our synthetic seismic training data, and we believe the CNN will be necessary to handle this greater complexity. So we plan to keep the CNN framework as we develop the data; this will be analyzed further in terms of future work in the Discussion section.

In Figure 12 we present the predicted brittleness values from the trained CNN vs the actual brittleness values in the testing data. These values are plotted across all 900 testing samples. We also present the distribution of these predictions and actual values in histograms in Figure 13. Note that the predicted values tend to be slightly larger in their spanned range than the actual values. However, a similar pattern of the predicted values is followed as shown in Figure 12 - this may indicate that the residual error is caused by some bias or static shift.

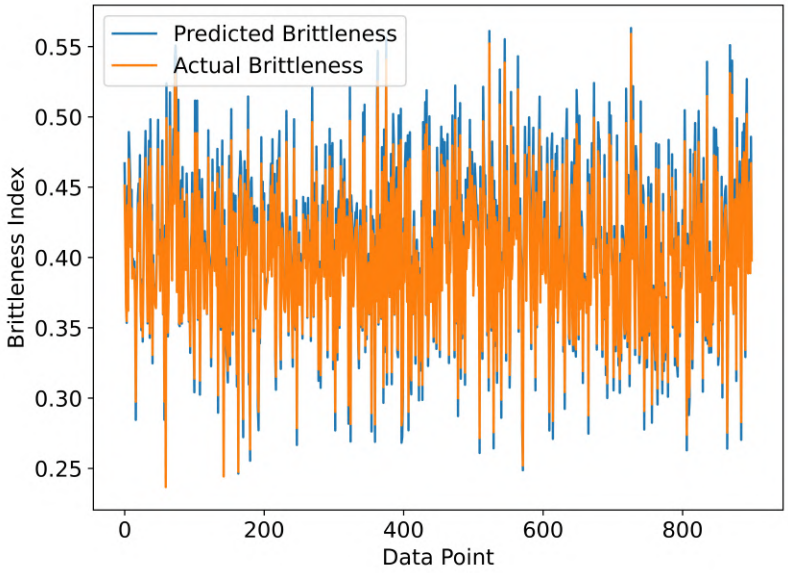

Figure 12: Predicted (blue) brittleness values from the CNN, vs actual (orange) brittleness values.

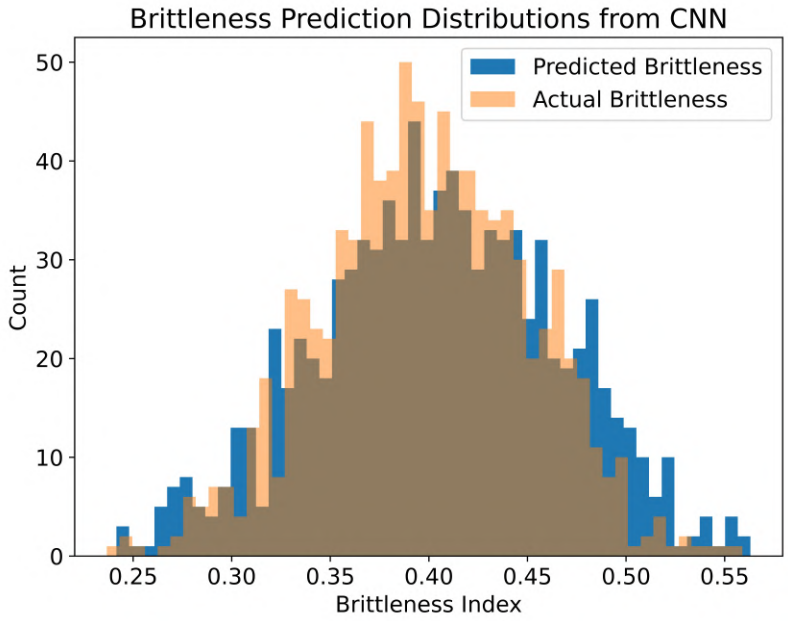

Figure 13: Distributions of predicted (blue) brittleness values from the CNN, vs actual (orange) brittleness values.

Field Data Results

We now test our trained CNN model on real seismic data collected in the Norwegian North Sea environment we emulated through our synthetic modeling. A 2D section of this seismic data is presented in Figure 14 below. This 2D section is approximately 1.5 kilometers deep by 1.8 kilometers across. The sampling in depth is at a 10 meter interval, while the sampling in offset is at a 12.5 meter interval. Each offset sample contains a recorded 1D seismic trace, which is the form of the data we trained on. When stacking these amplitudes across, they form a seismic image of the subsurface. The top of the Draupne Shale caprock is the black (positive amplitude) reflector, and the bottom of this layer is the white (negative amplitude) reflector, as indicated by the yellow arrows. We want to test how our trained model can predict brittleness values from the seismic amplitudes in this region.

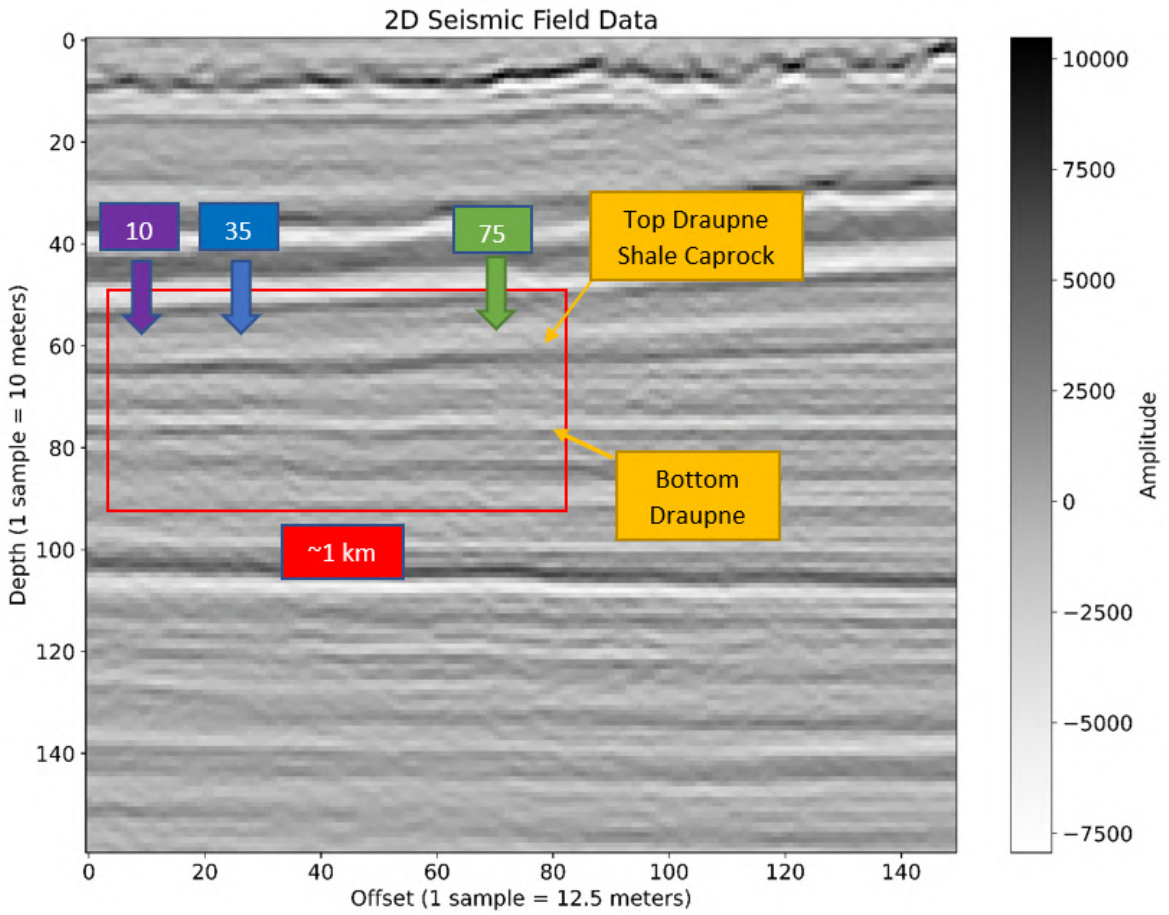

Figure 14: Real 2D seismic data collected in the North Sea, with the Draupne Shale Formation area boxed in red. The three offset locations that we extract traces from to conduct testing on are shown in the purple, blue, and green boxes.

To implement this, we extract three seismic traces at the offsets 10, 35, and 75. These correspond to offsets of 120 meters, 437.5 meters, and 937.5 meters, respectively. The traces are normalized by their maxima, and the seismic amplitudes more than 100 meters above and below the caprock are zeroed out, in the same manner as we did for the synthetic data. Then the trained CNN is used to predict the brittleness value of the caprock at these three offsets. The results are presented in Figure 15. For each offset we plot a reference synthetic that has a brittleness of 0.33. The offset 19 field seismic trace has a predicted brittleness of 0.338 - this is close to the synthetic's value and the amplitude correspondence is also similar. Moving to the offset 35 example, the predicted brittleness has increased to 0.351, and the amplitude of the field trace has decreased slightly. Finally on the offset 75 example, the predicted brittleness is significantly higher at 0.373, and the amplitudes have decreased significantly as well. This result is intuitive when compared to the synthetics shown in Figure 8, where we noted that the lower brittleness values corresponded to higher amplitudes for the top and base reflectors of the caprock. Most importantly, these values (on the order of 0.3 to 0.4) are close to the brittleness index values estimated from Rahman et al.âs more sophisticated mineralogical

brittleness model, and their elastic properties calculations (Rahman et al., 2022). This tells us that our model is predicting reasonable values. The predicted values are on the lower end of brittleness distributions, and thus this area would likely make a good CO2 storage site.

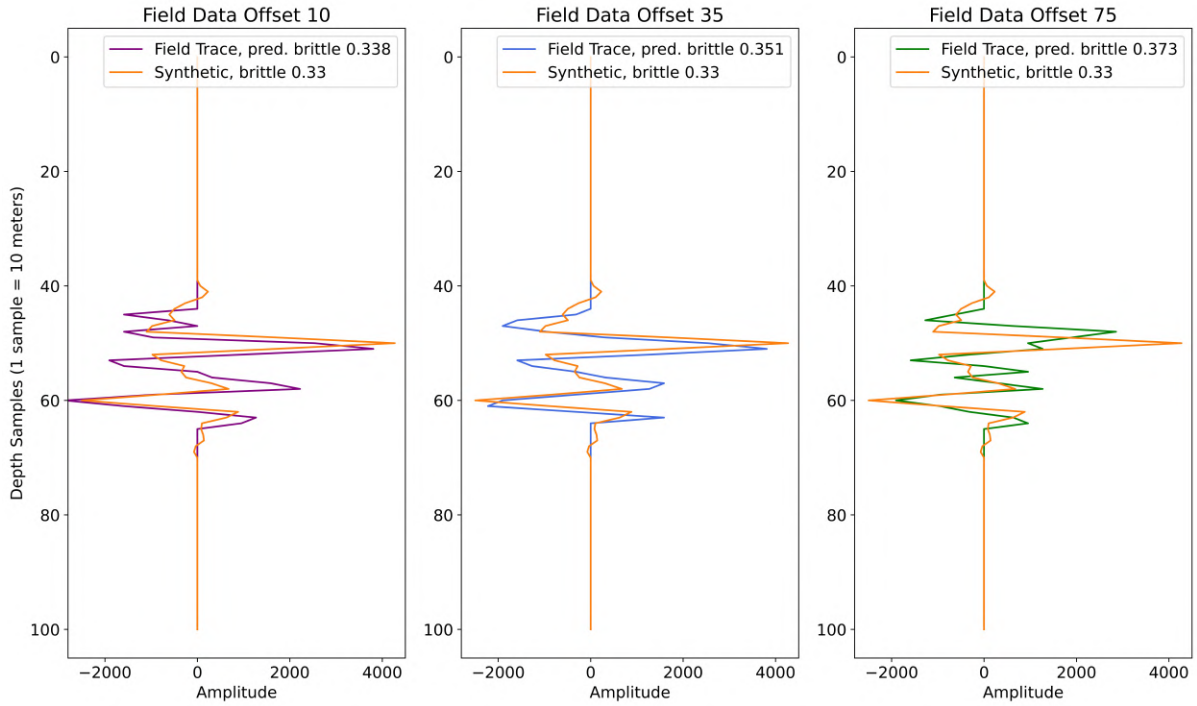

Figure 15: Real 2D seismic data collected in the North Sea, with the Draupne Shale Formation area boxed in red. The three offset locations that we extract traces from to conduct testing on are shown in the purple, blue, and green boxes.

Discussion

Our trained model was able to predict realistic brittleness values for a shale caprock in the North Sea. This is a step towards a very useful tool, because currently most methods for estimating shale brittleness require collecting samples of rock from wells and performing laboratory experiments. Well data of a shale formation is limited to a single location, and likely doesn't account for spatial heterogeneities in the rock properties. Therefore there could be significant variation in the brittleness of shale layers that is unknown and/or unaccounted for. A method that can operate on seismic data, which samples geologic properties much more densely then wells, would be able to provide more densely sampled estimations of shale brittleness as opposed to conventional methods, and perhaps identify spatial heterogeneites. This was the motivation behind our field data test in Figure 15 - to do a simple examination for spatial heterogeneity in the brittleness. Our model predicted that there was an increase in brittleness with offset moving along the shale layer, which could be very useful information in deciding where to inject CO2 in the subsurface. An operating company would want to inject in the area with the lowest shale brittleness so that fractures do not form as easily, and the CO2

cannot escape.

However it should be noted that there is significant work yet to be done in making our synthetic data and workflow more realistic. Note how in Figure 15 the shape of the synthetic data varies significantly from the field data. This is because our approximation of the seismic velocities and densities of the shale is not perfect. One way we could improve the synthetic data is by creating a more realistic mineralogical model that includes other clay minerals such as illite, as well as small quantities of feldspar, carbonates, pyrite, and organic content â similar to the more sophisticated model in Rahman et al., 2022.

Furthermore, we will in the future also aim to improve the seismic velocity and density estimations of the overburden and underburden rocks. We used a simple assumption of constant velocity and density for these rocks, but reality is more complex and heterogeneous than this. Therefore we would want to train a model over various seismic velocities and densities for these rocks, so that the model can learn more complex behavior. This is why we decided to stay with the CNN as our preferred model for testing and development - because we foresee increasing the complexity of our synthetic training data. A CNN would likely be better able to learn more complex patterns in the synthetic data, as these architectures have been often utilized for sophisticated signal and image analysis.

Conclusions

In this project we developed machine learning models to predict shale caprock brittleness from 1-dimensional seismic data. This has the application of determining whether a shale will fracture easily, and thus allow injected CO2 to escape in carbon sequestration operations. We used a mineralogical model of shale with clay content that varies according to normal distributions, and generated synthetic seismograms by using the velocities and densities corresponding to different values of the clay fraction. We used an empirical equation that maps the values of the clay fraction to shale brittleness, and thus created thousands of synthetics as training data with corresponding labels of brittleness value. It was found by examining our synthetics that lower seismic amplitudes corresponded to higher brittleness values, and vice versa. Three different machine learning models were trained on this data - a random forest regressor, a feed forward neural network, and a 1D convolutional neural network â and all three returned a similar level of error on the testing dataset, with mean squared errors of approximately 1.2e-4 and R2 values of approximately 0.97.

We tested the trained CNN on real seismic data imaging the Draupne Formation, a shale layer that acts as a caprock in the Norwegian North Sea. This 2D dataset was taken and three traces from different offsets were extracted to predict on, after being prepared in a similar manner to the training data. The predicted brittleness values ranged from 0.33 to 0.37 as offset increased along the seismic line, indicating possible heterogeneities in shale brittleness present in the formation. These values are in line with previous estimations of brittleness taken from previous mineralogical and elastic properties analyses using the same dataset. This result

is very promising, as it potentially provides accurate and spatially-dense estimations of shale brittleness compared to sparsely-sampled well data.

References

Aki, K., and P.G. Richards, 2002, "Quantitative Seismology," University Science Books.

Andreev, G.E., 1995, "Brittle Failure of Rock Materials," CRC Press.

Benson, S.M., and D.R. Cole, 2008, "CO2 Sequestration in Deep Sedimentary Formations," Elements, vol. 4, pg. 325-331, https://doi.org/10.2113/gselements.4.5.325.

Bourg, I.C., 2015, "Sealing Shales versus Brittle Shales: A Sharp Threshold in the Material Properties and Energy Technology Uses of Fine-Grained Sedimentary Rocks," Environmental Science and Technology Letters, vol. 2, 255-259, https://doi.org/10.1021/acs.estlett.5b00233.

Fawad, M., M.J. Rahman, N.H. Mondol, 2021, "Seismic reservoir characterization of potential CO2 storage reservoir sandstones in Smeaheia area, Northern North Sea," Journal of Petroleum Science and Engineering, vol. 205, 108812, https://doi.org/10.1016/j.petrol.2021.108812.

Gabrielsen, R.H., E. Skurtveit, and J.I. Faleide, 2021, "Caprock Integrity of the Draupne Formation, Ling Depression, North Sea, Norway," Norwegian Journal of Geology, vol. 100, 202019, https://dx.doi.org/10.17850/njg100-4-2.

Goodfellow, I., Y. Bengio, and A. Courville, 2016, "Deep Learning," MIT Press, https://www.deeplearningbook.org/.

James, G., D. Witten, T. Hastie, and R. Tibshirani, 2013, "An Introduction to Statistical Learning: with Applications in R," Springer, https://doi.org/10.1007/978-1-4614-7138-7.

Kim, Y., R. Hardisty, E. Torres, and K.J. Marfurt, 2018, "Seismic-facies classification using random forest algorithm," SEG Technical Program Expanded Abstracts, pg. 2161-2165, https://doi.org/10.1190/segam2018-2998553.1.

Kingma, D.P., and Ba, J., 2014, "Adam: A Method for Stochastic Optimization", https://doi.org/10.48550/arXiv.1412.6980.

Kiranyaz, S., O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, D.J. Inman, 2021, "1D convolutional

neural networks and applications: A survey," Mechanical Systems and Signal Processing, vol. 151, 107398, https://doi.org/10.1016/j.ymssp.2020.107398.

Mehta, P., M. Bukov, C.H. Wang, A.G.R. Day, C. Richardson, C.K. Fisher, and D.J. Schwab, 2019, "A High-Bias, Low-Variance Introduction to Machine Learning for Physicists," Physics Reports, vol. 810, pg. 1-124, https://doi.org/10.1016/j.physrep.2019.03.001.

Nielsen, M.A., 2015, "Neural Networks and Deep Learning," Determination Press, https://neuralnetworksanddeeplearning.com.

Rahman, M.J., J.R. Johnson, M. Fawad, N.H. Mondol, 2022, "Characterization of Upper Jurassic Organic-Rich Caprock Shales in the Norwegian Continental Shelf," Geosciences, vol. 12, 407, https://doi.org/10.3390/geosciences12110407.

Schroeder F., 2006, "Petroleum Geology and Geophysics: Lesson 6, Seismic to Well Ties," Exxon-Mobil, https://slideplayer.com/slide/703718/.

Shenfield, A., and M. Howarth, 2020, "A Novel Deep Learning Model for the Detection and Identification of Rolling Element-Bearing Faults," Sensors, vol. 20, 5112, https://doi.org/10.3390/s20185112.

Tarasov, B., and M. Randolph, 2011, "Superbrittleness of Rocks and Earthquake Activity," International Journal of Rock Mechanics and Mining Sciences, vol. 48, pg. 888-898, https://doi.org/10.1016/j.ijrmms.2011.06.013.

White, J.C., G.A. Williams, S. Grude, and R.A. Chadwick, 2015, "Utilizing Spectral Decomposition to Determine the Distribution of Injected CO2 at the SnÃ¸hvit Field," Geophysical Prospecting, vol. 63, pg. 1213-1223, https://doi.org/10.1111/1365-2478.12217.

Wu, X., L. Liang, Y. Shi, and S. Fomel, 2019, "FaultSeg3D: Using Synthetic Data Sets to Train an End-to-end Convolutional Neural Network for 3D Seismic Fault Segmentation," Geophysics, vol. 84, pg. IM35-IM45, https://doi.org/10.1190/geo2018-0646.1.

Wu, Y., Y. Lin, Z. Zhou, D.C. Bolton, J. Liu and P. Johnson, "DeepDetect: A Cascaded Region-Based Densely Connected Network for Seismic Event Detection," IEEE Transactions on Geoscience and Remote Sensing, vol. 57, no. 1, pp. 62-75, https://doi.org/10.1109/TGRS.2018.2852302.

Yang, L. and S.Z. Sun, 2020, "Seismic Horizon Tracking using a Deep Convolutional Neural Network," Journal of Petroleum Science and Engineering, vol. 187, 106709, https://doi.org/10.1016/j.petrol.2019.106709.

Yang, L., Y. Mao, D. Yang, Z. Han, S. Li, J. Cai, and M. He, 2022, "The Characteristic and Distribution of Shale Micro-Brittleness Based on Nanoindentation," Materials, vol. 15, 7143, https://doi.org/10.3390/ma1520714.

Appendix A: Analysis of bias-variance tradeoff of OLS, FFNN, and decision tree algorithms for a regression task

Summary

The bias-variance tradeoff analysis can be summarized as the study of the relationship between model complexity and the development of the mean square error (MSE). The MSE can be re-written as the sum the squared bias of model $\tilde{\mathbf{y}}$, the variance of $\tilde{\mathbf{y}}$, and the variance of the noise σ^2:

$$Bias[\tilde{\boldsymbol{y}}]^2 + Var[\tilde{\boldsymbol{y}}] + \sigma^2 \tag{A1}$$

In this exercise, we perform the bias-variance tradeoff analysis for three supervised machine learning algorithms (i) Ordinary Least Square (ii) Feed-Forward Neural Network (iii) Decision tree. Those algorithms are applied to find a model that fits 100 data points generated by the Franke Function, a two-dimensional function. Here, we use the bootstrap method as a re-sampling technique. The objective is to analyze the pros and cons of each method by decomposing the MSE into bias and variance.

Data and Method

Ordinary Least Squares (OLS)

OLS is a linear regression that assume that there exists a convex function that can approximate our data. The solution of the linear regression problem is the optimal predictor β that minimizes the cost function $C(\mathbf{X}, \beta)$:

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \frac{1}{n}[(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})] = \frac{1}{n}\left\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right\|_2^2 \tag{A2}$$

Where $\mathbf{X}$ is the design matrix, $\mathbf{y}$ the data vector, and $\mathbf{n}$ the number of data points. The solution of this linear algebra problem can be conveniently expressed analytically, provided that the Hessian matrix $\mathbf{X^T X}$ is invertible:

$$\boldsymbol{\beta} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{A3}$$

Decision Trees

A decision tree is defined by a root node, the interior nodes, and the final leaf nodes. The nodes are connected by branches. A node specifies a test of some attribute of the incoming instance. The overarching concept of decision trees is a top-down approach where the leaf nodes contain the predictions that we will compare to our target value (regression) or label (classification). For regression tasks, a decision tree is grown as follows:

1. We find a cut point s and split the data space into 2 non-overlapping quadrangular regions that minimizes the MSE. The prediction is the mean of the response values in each region.
2. We repeat the process and subdivide the two previously identified regions according to an optimal cut point that minimizes the MSE.
3. The process continues until a stopping criterion is reached. In our case, we define a maximum depth of the tree.

Feed Forward Neural Networks

The reader is referred to the Theory section of the main report.

Bootstrap Resampling

The Bootstrap method is a re-sampling technique underlain by the fundamental central limit theorem. The theorem states that the probability-density function (PDF) P(x) of the average of m random values with a corresponding PDF p(x) is a normal distribution whose variance is the variance of p(x) divided by m, the number of experiments. For each bootstrap, we draw with replacements a training set with indexes following P(x). The optimal predictor is then determined under the chosen training set.

Franke Function

The Franke function is a function of two variables where $(x, y)\epsilon[0, 1]$ and is expressed as follows:

$$
\begin{aligned}
\tilde{\boldsymbol{z}} &= \boldsymbol{X}\boldsymbol{\beta} \\
\tilde{z}_0 &= \beta_0 + \beta_1 x_0 + \beta_2 y_0 + \beta_3 x_0^2 + \beta_4 x_0 y_0 + \beta_5 y_0^2 + \ldots + \beta_{p-1} y_0^m, \\
\tilde{z}_1 &= \beta_0 + \beta_1 x_1 + \beta_2 y_1 + \beta_3 x_1^2 + \beta_4 x_1 y_1 + \beta_5 y_1^2 + \ldots + \beta_{p-1} y_1^m, \\
&\vdots \\
\tilde{z}_n &= \beta_0 + \beta_1 x_n + \beta_2 y_n + \beta_3 x_n^2 + \beta_4 x_n y_n + \beta_5 y_n^2 + \ldots + \beta_{p-1} y_n^m,
\end{aligned}
\tag{A4}
$$

The Franke Function is depicted in Fig. A1 for two cases, with and without normally distributed noise. We will use a normally distributed noise with a variance of 0.1 in this exercise.

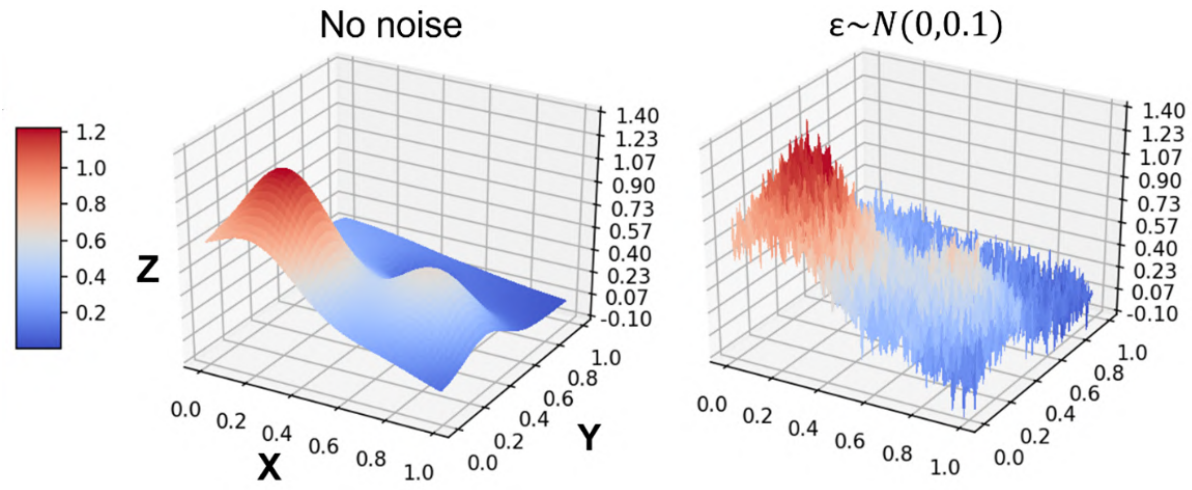

Figure A1: 3D visualization of the Franke Function without added noise (left panel) and with normally distributed noise with variance 0.1 and mean 0 (right panel).

Results

The results of the bias-variance tradeoff analysis for the bootstrapped OLS, FFNN and regression tree are presented in Fig. A2. A particular attention was given on how to represent model complexity for each learning techniques.

OLS

For linear regression, the model complexity is defined by the degree of the polynomial function that is used to fit the observations. On Fig. A2, one can clearly see that the optimal model is a polynomial of degree 4 giving a minimal prediction error of 0.02 as well as a low variance and bias. For degree beyond 4, the prediction error explodes, mainly driven by a high variance, and thereby reaching the overfitting domain.

FFNN

On Fig. A2, we observe that a FFNN with a single or two hidden layers containing 10 nodes are optimal to fit the data, with a prediction error of 0.02. If we increase further the depth of the network (model complexity), we observe that the MSE explodes, with the model bias being the dominant factor. The bias shows the highest contribution to overfitting for FFNN.

Decision Tree

For the decision tree, we observe that a minimal MSE of 0.035 is obtained for a tree depth (model complexity) of 5. Beyond this depth, we observe a stabilization of both bias and variance values for all regression trees to respectively 0.02 and 0.015.

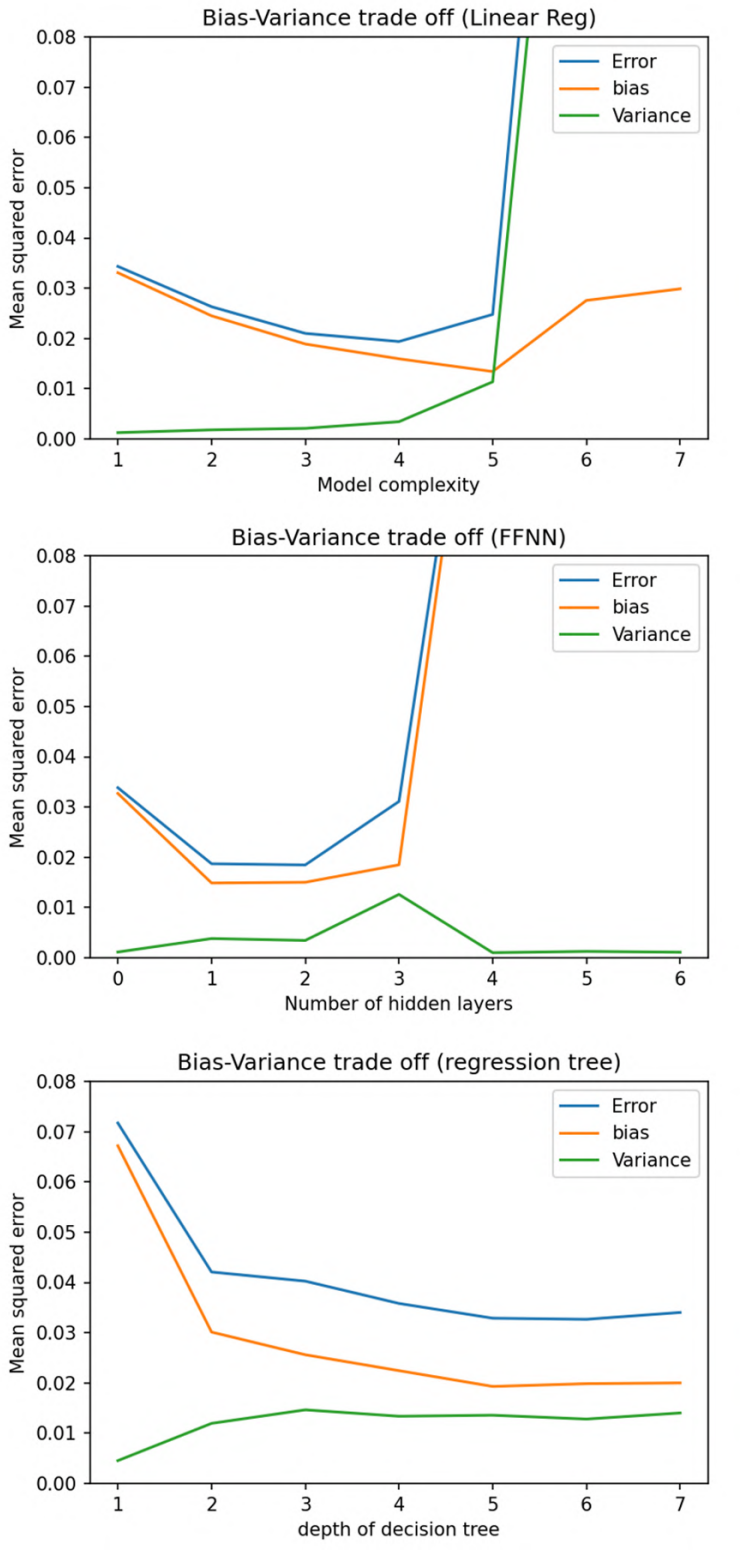

Figure A2: Bias-variance tradeoff results for three supervised machine learning algorithms using n=100 data points generated by the Franke Function (Top) Ordinary least square (middle) Feed-Forward Neural Network (bottom) Decision tree

Discussion and Conclusions

The three algorithms used in this exercise have distinct characteristics in terms bias-variance tradeoff. The OLS shows an error dominated by high variance when dealing with overfitting models whereas FFNN shows a dominant bias contribution for overly deep learning networks. In turn, regression tree shows a stable contribution of both bias and variance for increasing model complexity. Here, decision tree appears to be more robust to overfitting than the previous two learning methods. However, the overall predictive power of regression tree underperforms compared to the two models from FFNN and OLS for this regression problem. Following Occamâs razor principle, we conclude the OLS is the most adapted algorithm for this regression problem. We foresee that the robustness of decision tree to overfitting may be relevant for more complex and noisy datasets.