**UNIVERSITÀ DEGLI STUDI DI PADOVA**
DIPARTIMENTO DI INGEGNERIA CIVILE, EDILE E AMBIENTALE
*Department of Civil, Environmental and Architectural Engineering*

**COLUMBIA UNIVERSITY**
FU FOUNDATION SCHOOL OF ENGINEERING AND APPLIED SCIENCE
*Department of Civil Engineering and Engineering Mechanics*

Corso di Laurea Magistrale in Ingegneria Civile

**TESI DI LAUREA**

# MACHINE LEARNING APPROACH TO DATA-DRIVEN MULTISCALE TRACTION-SEPARATION LAWS FOR GRANULAR MATERIALS

Relatore:
Chiar.ma PROF.SSA VALENTINA SALOMONI

Laureanda: SARA MICHIELETTO
1179043

**ANNO ACCADEMICO 2018-2019**

# Contents

# List of Figures

## List of Tables

# Acknowledgments

First, I would like to express my most gratitude to my advisor Professor Valentina Salomoni for the precious opportunity she presented me and the inspirational encouragement throughout this experience.

I am grateful to Professor Steve WaiChing Sun for the precious time and work spent helping me understand the object of this dissertation and having me as part of his staff for a short period at Columbia University.

During the last five years at University, I enjoyed support from a group of my colleagues. With our knowledge and friendship we have help each other during these years and this meant a lot to me. We have shared a lot of unforgettable moments and values such as true friendship and hard work.

Finally, my sincere thanks are given to my boyfriend Filippo, my brother Matteo and my parents, Laura and Danilo, who always love and encourage me. Without their support, I could not have accomplished this experience.

# 1. Introduction

Machine learning is considered a specific field of artificial intelligence which has the capability to learn from data. This is the reason why it is so commonly used in very different fields of science, finance and industry nowadays.

In this work granular materials properties are used to learn and predict their constitutive laws. In particular, Traction-separation laws are often highly simplistic due to the difficulty to propose a proper model that captures the phenomenology. By incorporating the micro-structural information via a Neural Network, more realistic and complex constitutive laws can be generated automatically.

The goal of this thesis is to develop a Recurrent Neural Network that could predict the constitutive laws for granular materials that are implemented. Different information from multiple sub-scales can be used sequentially to generate macroscopic prediction with a low computational and time cost.

The entire dataset is given by DEM simulations for granular materials but it can be adjusted for all kind of materials. From rocks to sand, from concrete to steel.

The constitutive laws (traction-separation laws) obtained from homogenizing the DEM responses are used as the data set for training and validating the neural network models.

The code written in Python, trains and predicts the constitutive relationship that depends on de dataset built in the previous simulations. The key of this thesis is to evaluate the neural network in terms of error and loss. These parameters help to understand if the model is predicting well the curves.

As for the implementation, the model was written with Keras, a high-level Python deep learning library, to build the neural networks and complete the training procedure. This model-level library allows for easy and fast prototyping of machine learning models.

Three types of neural network have been developed, the Dense, LSTM and GRU. Differences between the models as well as their performances will be presented further in this work.

# 2. Elements of Machine Learning

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples provided. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

AI and machine learning algorithms aren't new. The field of AI dates back to the 1950s. Arthur Lee Samuels, an IBM researcher, developed one of the earliest machine learning programs — a self-learning program for playing checkers. In fact, he coined the term machine learning. His approach to machine learning was explained in a paper published in the IBM Journal of Research and Development in 1959.

Over the decades, AI techniques have been widely used as a method of improving the performance of underlying code. In the last few years with the focus on distributed computing models and cheaper compute and storage, there has been an increase of interest in AI and machine learning that has led to a huge amount of money being invested in start-up software companies.

Tom M. Mitchell, computer scientist at Carnegie Mellon University, introduced a definition of Machine learning as:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Therefore, in general learning is about improving future performance using past experience, reducing as more as possible human intervention or assistance. The main three criteria about the machine learning in solving a problem are:

- It is possible to recognize a pattern;
- It isn’t possible to find a way to describe it mathematically;
- There are data that represents the pattern.

Machine learning tasks are usually classified in four different wide categories, depending on the nature of the problem faced as in Fig 2.1:

- Supervised learning: through an unknown target function $y = f(x)$ it is possible to map the input x to output y.
- Unsupervised learning: the input given is not labelled and the goal of the algorithm is to infer a function to describe hidden structure or pattern in the input.
- Reinforcement Learning: the input and output do not need to be labelled. The goal is to find a balance between exploration (of uncharted territory) and exploitation (of current knowledge).
- Deep learning: it can be implemented both as supervised or unsupervised technique with the development of neural network.

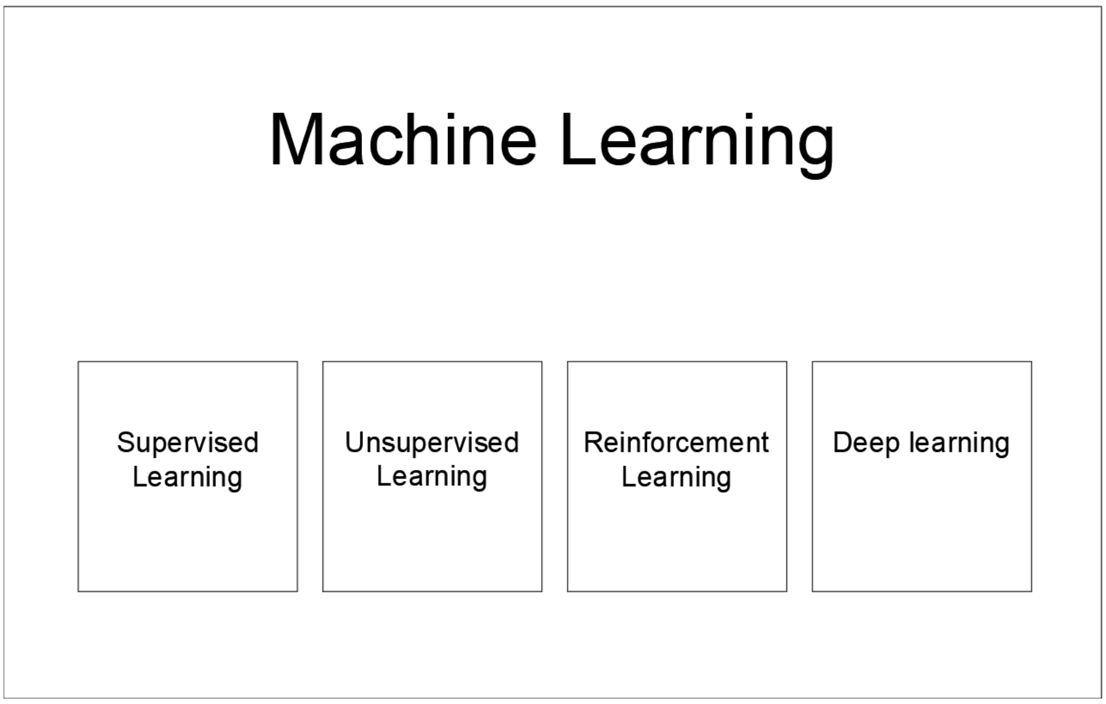


*Figure 2.1 Machine learning categories*

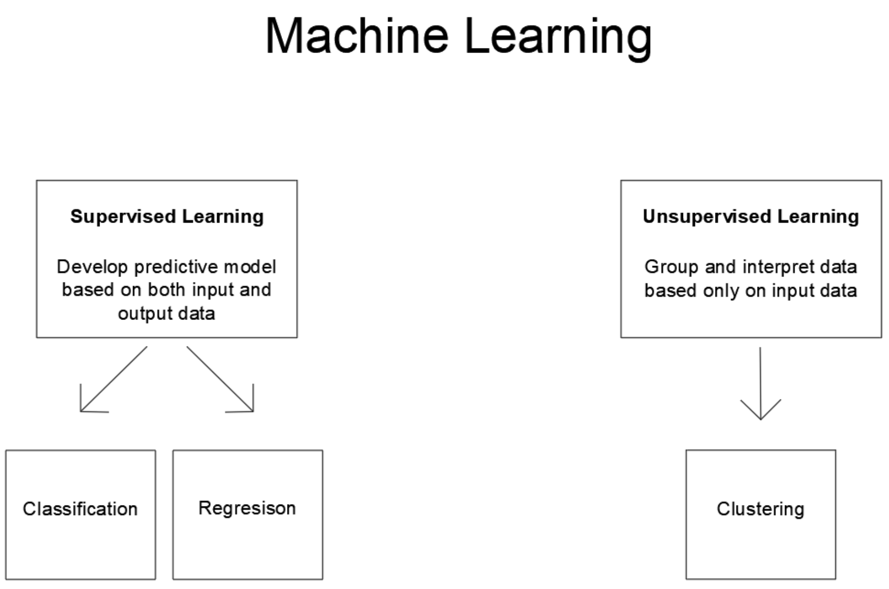

Machine Learning
Supervised Learning
Develop predictive model based on both input and output data
Unsupervised Learning
Group and interpret data based only on input data
Classification
Regresison
Clustering

*Figure 2.2 Scheme of Machine learning approaches*

The problem faced in this thesis required a supervised learning approach, therefore only this branch of machine learning will be discussed.

## 2.1. Supervised learning

Supervised learning typically begins with an established set of data and a certain understanding of how those data are classified. Supervised learning is intended to find patterns that can be applied to any analytics process. A dataset must be labelled with features that define the meaning of data.

The goal of supervised learning is to use the inputs to predict the values of the outputs. A set of variables might be denoted as inputs and they can be measured or pre-set. These have some influence on one or more outputs.

In the statistical literature the inputs are often called the predictors and more classically the independent variables. The outputs are called the responses, or classically the dependent variables.

The aim of supervised machine learning is to build a model that makes predictions based on evidence in the presence of uncertainty. A supervised learning algorithm takes a known set of input data and known responses to the data (output) and trains a model to generate reasonable predictions for the response to new data.

There are two types of Supervised Learning techniques: Regression and Classification. Classification separates the data, Regression fits the data.

- Classification is a technique that aims to reproduce class assignments. It can predict the response value and the data is separated into "classes".
- Regression is a technique that aims to reproduce the output value. Regression techniques predict continuous responses. Typical applications include electricity load forecasting and algorithmic trading.

Starting from a set of examples the algorithm is guided to describe a model able to predict the correct output. At this point the prediction model must be validated with another known dataset independent from the training set. Only when the validation phase is satisfactory the algorithm can be considered reliable for use on unknown data.

Components of learning:

- Input: **x**
- Output: **y**
- Target function: $\mathbf{f: X \rightarrow Y}$ where X is a set of all inputs and Y set of all outputs. The target function is initially unknown.
- Data: $\mathbf{(x_1,y_1), (x_2 ,y_2),\dots,(x_n,y_n)}$
- Hypothesis: $\mathbf{g: X \rightarrow Y}$ is the formula that approximate the target function. It is a known function and the goal is g to approximate well the target function f.

The learning algorithm is the step between the training examples of data and the final hypothesis **g**. It chooses a formula for the final hypothesis between a set of candidate formulas which is called Hypothesis Set ***H***.

The learning model is formed by the hypothesis set and the learning algorithm.

The Hypothesis set is composed by a number of sets that can be chosen to be the final hypothesis. The final hypothesis will be the function that will approximate in the best way the unknown target function. Therefore, the mathematically definition of the hypothesis set is:

$$H = \{h\} \ , g \in H$$

## 2.2. Perceptron

The perceptron is one of the simplest algorithms for supervised learning, introduced by Rosenblatt in 1958. The perceptron can be considered the simplest Artificial Neural Network algorithm because it is composed just by one artificial neuron. It consists in binary classifications through a threshold function.

A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and 0 otherwise. In figure 2.3 it is possible to see the scheme of a single layer perceptron,

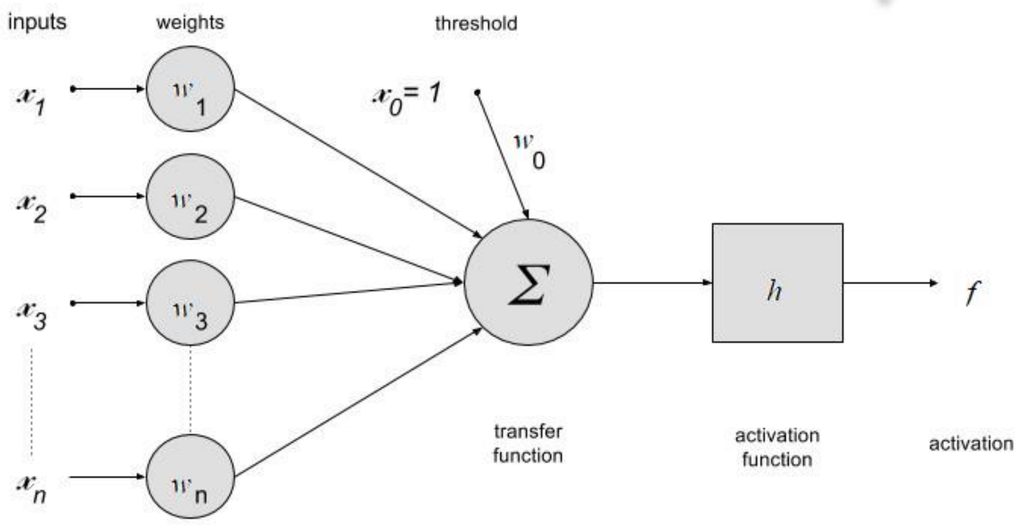


*Figure 2.3 Perceptron Scheme*

This function maps its input x to an output value f(x):

$$f(x) = \begin{cases} 1 \ if \ w \cdot x + b > 0 \\ 0 \ otherwise \end{cases}$$

Where w is a vector of real-valued weights that determines the contribution of input x to the perceptron output and b is the bias.

Linear models use the ‘signal’ as a linear sum described below:

$$w^T x = \sum_{i=1}^{d} w_i x_i$$

The number of inputs to the perceptron is d. Learning a perceptron means choosing values for the weights. It is possible to have a classification linear system like the perceptron that uses the signal and uses the sign of it to make a binary decision.

$$h(x) = sign(w^T x)$$

Regression takes real values and uses them as outputs:

$$h(x) = w^T x$$

So the simplest algorithm is the linear regression in which it puts the inputs in a particular matrix form so it can give the optimal value of the weight vector.

$$w = (X^T X)^{-1} X^T y$$

The perceptron algorithm is also termed the single-layer perceptron, to distinguish it from a multilayer perceptron, which is a misnomer for a more complicated neural network.

## 2.3. Deep learning and Artificial Neural Network

A neural network attempts to mimic the way a human brain approaches problems and uses layers of interconnected units to learn and infer relationships based on observed data. A neural network can have several connected layers. When there is more than one hidden layer in a neural network, it is sometimes called deep learning. Neural network models are able to adjust and learn as data changes. Neural networks are often used when data is unlabelled or unstructured.

The model was created with biological inspiration in order to replicate the human ability of learning. The structure of biological neural network are neurons and synapsis. Artificial Neural Network retained the biological concept of artificial neurons, which receive input, combine the input with their internal state (activation) and an optional threshold using an activation function, and produce output using an output function

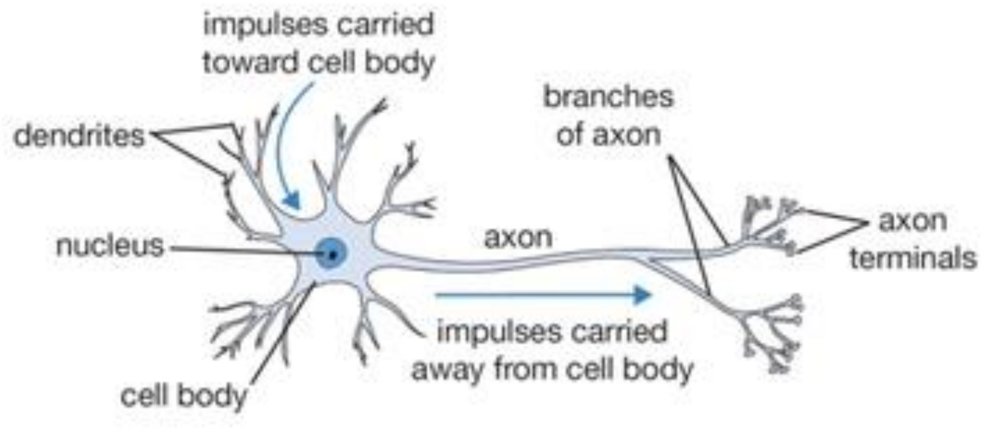

impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

*Figure 2.4Biological Neuron*

The term ‘neural network’ has its origins in 1943 with McCulloch and Pitts in attempts to find mathematical representations of information processing in biological systems.

ANNs began as an attempt to exploit the architecture of the human brain to perform tasks with specific algorithms.

An artificial neural network is composed by nodes. A node, also called a neuron or Perceptron, is a computational unit that has one or more weighted input connections, a transfer function that combines the inputs in some way, and an output connection.

Nodes are then organized into layers to comprise a network. A single-layer artificial neural network, also called a single-layer, has a single layer of nodes, as its name suggests. Each node in the single layer connects directly to an input variable and contributes to an output variable.

Neurons are connected to each other in various patterns, to allow the output of some neurons to become the input of others. The network forms a directed, weighted graph:

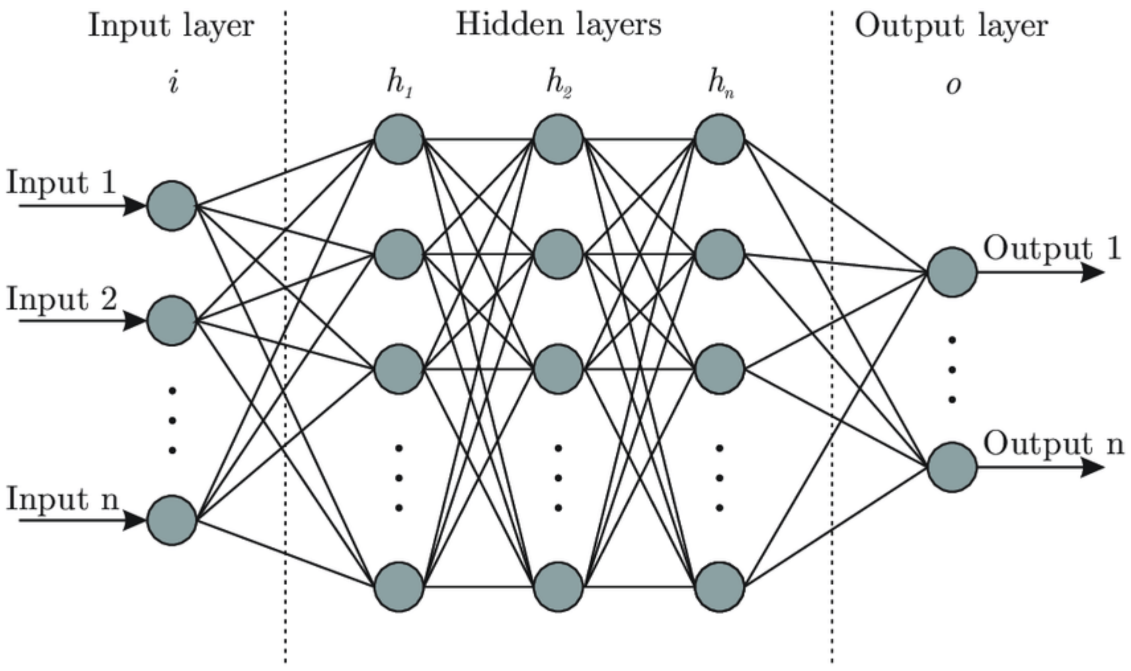


*Figure 2.5 Artificial Neural Network*

An Artificial neural network consists in large number of neurons, each of one is a single perceptron. It is possible to model each neuron as a function that sum the inputs with their weight and add a bias. The output will be based on an activation function.

The activation function is a function which maps the arbitrary output of the logit function to any specific range of values. It’s usually used to add some non-linearity to our model. This allows the network to combine the inputs in more complex ways and in turn provide a richer capability in the functions they can model. The bias decides when a neuron stays inactive or in other words, it decided how high the weighted sum needs to be for the neuron to be meaningfully active.

Deep learning is a specific method of machine learning that incorporates neural networks in successive layers in order to learn from data in an iterative manner. Deep learning is especially useful when trying to learn patterns from unstructured data.

Deep learning is in other words, a complex neural network that are designed to emulate how the human brain works in order to train computers to deal with abstractions.

Deep learning algorithms are improved versions of artificial neural networks algorithms. They use multiple layers of artificial neural networks to model the way the human brain processes things like light and sound into vision and hearing. In general, deep learning algorithms are built off of unsupervised learning run on multiple levels of the data.

They are concerned with building much larger and more complex neural networks and, as commented on above, many methods are concerned with very large datasets of labelled analogue data, such as image, text. audio, and video.

The most popular deep learning algorithms are:

- Convolutional Neural Network (CNN)
- Recurrent Neural Networks (RNNs)
- Long Short-Term Memory Networks (LSTMs)
- Stacked Auto-Encoders
- Deep Boltzmann Machine (DBM)
- Deep Belief Networks (DBN)

Neural networks and deep learning are often used in image recognition, speech, and computer vision applications. A neural network consists of three or more layers: an input layer, one or many hidden layers, and an output layer as seen in Fig.2.5.

The dataset is ingested through the input layer. Then the data is modified in the hidden layer and the output layers based on the weights applied to these nodes. The typical neural network may consist of thousands or even millions of simple processing nodes that are densely interconnected. The term deep learning is used when there are multiple hidden layers within a neural network. Using an iterative approach, a neural network continuously adjusts and makes inferences until a specific stopping point is reached. Deep learning is a machine learning technique that uses hierarchical neural networks to learn from a combination of unsupervised and supervised algorithms. Deep learning is often called a sub-discipline of machine learning. Typically, deep learning learns from unlabelled and unstructured data. While deep learning is very similar to a traditional neural network, it will have many more hidden layers. The more complex the problem, the more hidden layers there will be in the model.

## 2.4. Structure of the network

The network is a composite function of multiple neurons in the form of layers.

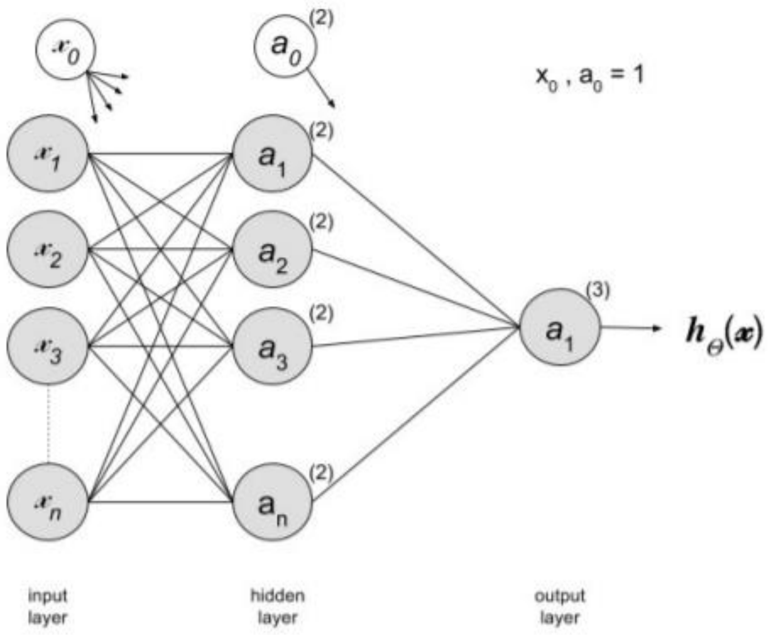


*Figure 2.6 Three layer neural network*

A typical neural network consists of 3 types of layers as seen in Fig. 2.6.:

- The input layer: The given data points are fed into this layer. There can be only 1 input layer. The number of neurons in this layer is equal to the number of inputs.
- The hidden layers: These are the layers which try to find patterns in the inputs to get the outputs we need. A network can have any number of hidden layers. Nodes of this layer are not exposed to the outer world, they are the

part of the abstraction provided by any neural network. Hidden layer performs all sort of computation on the features entered through the input layer and transfer the result to the output layer.

- The output layer: It is the last layer of neurons that produces given outputs for the program also known as the predictions of the network. The number of neurons in this layer is equal to the number of values need to be predicted.

A traditional artificial neuron is composed of some weighted inputs, a transformation function and activation function corresponding to the biological neuron's axon.

For each activation $a_i^{(2)}$ in the second layer an independent vector of weight $\Theta_i^{(1)}$ is used. Therefore, it is possible to write:

$$a_i^{(2)} = \Theta_i^{(1)} \cdot x$$

Where $\Theta_i^{(1)}$is the row I of the matrix$\Theta^{(1)}$ that maps layer 1 to layer 2.

### 2.4.1. Training the network

It is possible to describe the relationship between the input variables and the output variables as a complex mathematical function. For a given model problem, there must exist a true mapping function that can properly map input variables to output variables. Each neuron of the network has a unique set of weights and biases so it is possible to train the network by using a general algorithm that optimize the function.

The training phase is probably the most important one, as the final performances depend on the predictive model built.

The dataset must be as more representative as possible of the task of the program. The aim of this phase is trying to build a model able to fit the data provided, that is predict the correct output for each input provided as best as possible.

Training a deep learning neural network model using stochastic gradient descent with backpropagation involves choosing a number of components and hyperparameters. An error function must be chosen, often called the objective function, cost function, or the loss function. The loss function is used to estimate the performance of a model with a specific set of weights on examples from the training dataset.

The Gradient descent algorithm calculates the gradients of the function at a current point that is the direction that minimize the function.

The search or optimization process requires a starting point from which to begin model updates. The starting point is defined by the initial model parameters or weights. Because the error surface is non-convex, the optimization algorithm is sensitive to the initial starting point. As such, small random values are chosen as the initial model weights, although different techniques can be used to select the scale and distribution of these values. These techniques are referred to as “weight initialization”.

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).It is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm.

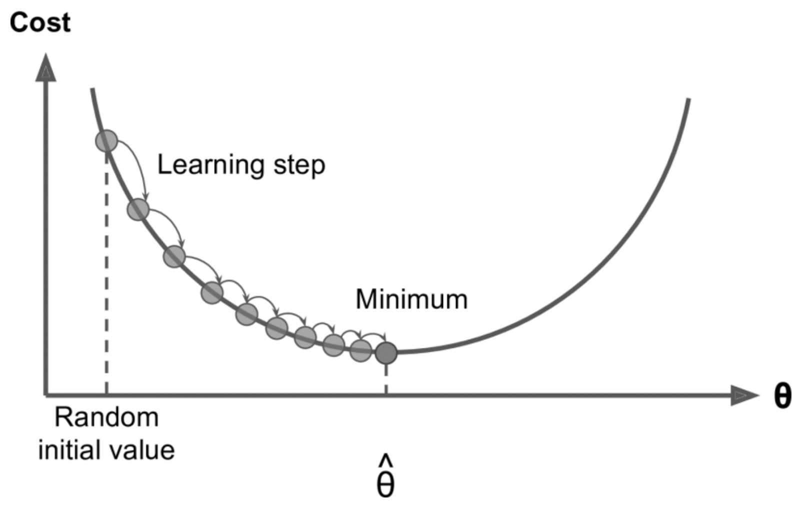


*Figure 2.7 Gradient Loss Algorithm scheme*

Gradient Descent is a very generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.

An important parameter in Gradient Descent is the size of the steps, determined by the learning rate hyperparameter. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time.

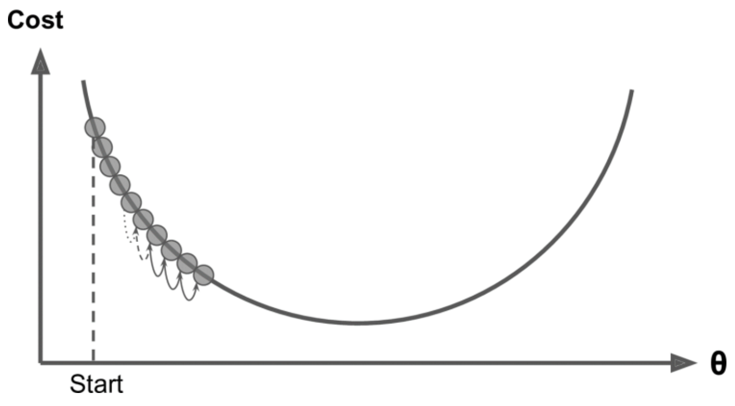


*Figure 2.8 Learning Rate*

Neural networks are trained using stochastic gradient descent and require the choice of a loss function when designing and configuring your model. In this thesis the mean squared error is used as loss function as it is also the default function

- Mean Squared Error:

  Mean squared error is calculated as the average of the squared differences between the predicted and actual values. The result is always positive regardless of the sign of the predicted and actual values and a perfect value is 0.0. The squaring means that larger mistakes result in more error than smaller mistakes, meaning that the model is punished for making larger mistakes.

$$loss = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - y_i^p\right)^2$$

### 2.4.2. Testing the network

Also known as Validation phase, its aim is to test the performances of the prediction model during the training. For this phase it is usually used a particular dataset called the test set which is composed by either input and outputs. This set of examples should be independent from the testing set. In all the model of this thesis, the 10% of the data are used as validation data.

During the testing the output are predicted but the algorithm does not use them to improve its performances.

The performances are evaluated comparing the differences between the output of the training set examples and the testing results.

### 2.4.3. Activation function

Activation function decides whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

As seen in Fig. 2.6, the activation function is used for each output giving

$$a_i^{(2)} = \varphi(\theta_{i0}^{(1)}x_0, \theta_{i1}^{(1)}x_1, \theta_{i2}^{(1)}x_2, \theta_{i3}^{(1)}x_3, \theta_{i4}^{(1)}x_4)$$

or, in a compact way:

$$a^{(2)} = \varphi(\theta^{(1)} \cdot x)$$

Each activation $a_i^{(2)}$ is then mapped to $a_1^{(3)}$ through a second weight matrix $\theta^{(2)}$. In other words, it is possible to map the input x directly to the output $a_1^{(3)}$ using the notation $a_1^{(3)} = h_\Theta(x)$ where $\Theta = \{\theta^{(1)}, \theta^{(2)}\}$.

Neural networks have neurons that work in correspondence of weight, bias and their respective activation function. In a neural network, the weights must the updated along with the biases of the neurons. The process back-propagation consists in the update the weights and bias depending on the error at the output.

Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

A neural network is comprised of layers of nodes and learns to map examples of inputs to outputs. For a given node, the inputs are multiplied by the weights in a node and summed together. This value is referred to as the summed activation of the node. The summed activation is then transformed via an activation function and defines the specific output or "activation" of the node.

**Linear function**

The simplest activation function is referred to as the linear activation, where no transform is applied at all. A network comprised of only linear activation functions is very easy to train but cannot learn complex mapping functions. Linear activation functions are still used in the output layer for networks that predict a quantity (e.g. regression problems).

$$f(x) = x$$

**Non-linear functions**

Nonlinear activation functions are preferred as they allow the nodes to learn more complex structures in the data. Traditionally, two widely used nonlinear activation functions are the sigmoid and hyperbolic tangent activation functions.

- Sigmoid function:

  It is a non-linear function also called as logistic function. It is usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1. The shape of the function for all possible inputs is an S-shape from zero up through 0.5 to 1.0. For a long time, through the early 1990s, it was the default activation used on neural networks.

$$f(x) = \sigma(x) = \frac{1}{(1 + e^{-x})}$$

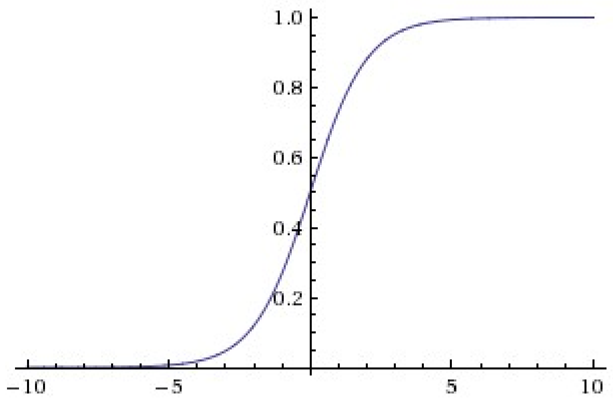


*Figure 2.9 Sigmoid Function*

- Tanh function:

  It is also known as Tangent Hyperbolic function, it is non-linear and it is generally used in hidden layers of neural network because its values range between -1 to 1. In the later 1990s and through the 2000s, the tanh function was preferred over the sigmoid activation function as models that used it were easier to train and often had better predictive performance.

$$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$

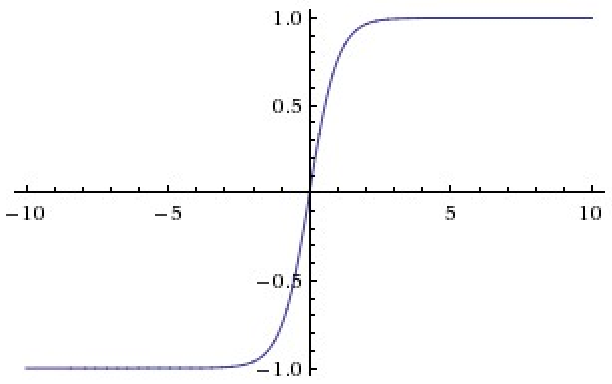


*Figure 2.10 Tangent Hyperbolic Function*

A general problem with both the sigmoid and tanh functions is that they saturate. This means that large values snap to 1.0 and small values snap to -1 or 0 for tanh and sigmoid respectively. Further, the functions are only really sensitive to changes around their mid-point of their input, such as 0.5 for sigmoid and 0.0 for tanh.

The limited sensitivity and saturation of the function happen regardless of whether the summed activation from the node provided as input contains useful information or not. Once saturated, it becomes challenging for the learning algorithm to continue to adapt the weights to improve the performance of the model.

Layers deep in large networks using these nonlinear activation functions fail to receive useful gradient information. Error is back propagated through the network and used to update the weights. The amount of error decreases dramatically with each additional layer through which it is propagated, given the derivative of the chosen activation function. This is called the vanishing gradient problem and prevents deep (multi-layered) networks from learning effectively.

In order to use stochastic gradient descent with backpropagation of errors to train deep neural networks, an activation function is needed that looks and acts like a linear function, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned such as the RELU function.

- RELU

  The Rectified linear unit is the most used non-linear function in Artificial Neural Networks. It is implemented especially in hidden layers because its computational cost is lesser then the previous functions. The rectified linear activation function is a piecewise linear function that will output the input directly if is positive, otherwise, it will output zero. The function is linear for values greater than zero, meaning it has a lot of the desirable properties of a linear activation function when training a neural network using backpropagation. Yet, it is a nonlinear function as negative values are always output as zero.

$$f(x) = \begin{cases} 0 \; for \; x \leq 0 \\ x \; for \; x > 0 \end{cases}$$

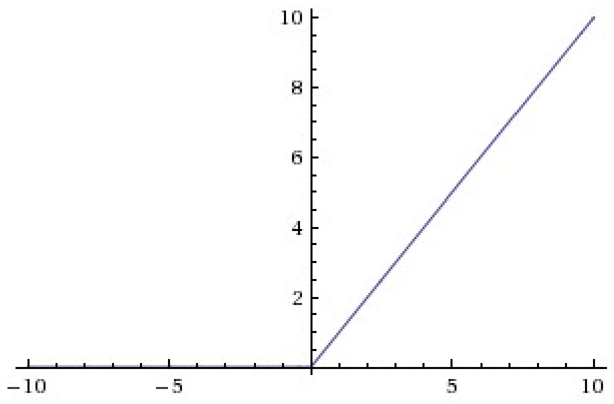


*Figure 2.11 RELU Function*

In the last chapter of this thesis, there will be presented differences in terms of train and validation loss between the sigmoid and the Relu function.

### 2.4.4. Back propagation algorithm

Backpropagation refers to a technique from calculus to calculate the derivative (e.g. the slope or the gradient) of the model error for specific model parameters, allowing model weights to be updated to move down the gradient. As such, the algorithm used to train neural networks is also often referred to as simply backpropagation.

Backpropagation is a short form for "backward propagation of errors." It is a standard method of training artificial neural networks. In 1961, the basics concept of continuous backpropagation was derived in the context of control theory by J. Kelly, Henry Arthur, and E. Bryson.

This method allows the network to modify the way how the steps are computed making possible to adjust the output. In fact, it takes the error associated with a wrong guess by a neural network and uses that error to adjust the neural network’s parameters in the direction of less error.

Recalling the general structure of supervised learning, this is done through the training data in the training step: knowing the output related to each input is possible to evaluate the error with the respect to the output of the network. The backpropagation step is the algorithm that allows to change the network trying to reduce the computed error.

Backpropagation computes the gradient in weight space of a feedforward neural network, with respect to a loss function.

Backpropagation simplifies the network structure by elements weighted links that have the least effect on the trained network. It helps to assess the impact that a given input variable has on a network output. The knowledge gained from this analysis should be represented in rules. It takes advantage of the chain and power rules allows backpropagation to function with any number of outputs.

**General Algorithm**

Given the following term definitions, it is possible to proceed to the explanation of the algorithm phases.

$w^k_{ij}$= weight for node j in layer $l_k$ for incoming node i

$b^k_i$=bias for node i in layer $l_k$

$a^k_i$=product sum plus bias (activation) for node i in layer $l_k$

$o^k_i$=output for node i in layer $l_k$

$r_k$= number of nodes in layer $l_k$

g= activation function for the hidden layer nodes

$g_o$= activation function for the output layer nodes.

The backpropagation algorithm proceeds in the following steps, assuming a suitable learning rate $\alpha$ and random initialization of the parameters $w_{ij}{}^k$:

1) Forward phase: it consists in the calculation for each input-output pair $(\overrightarrow{x_d}, y_d)$ of the parameters $\widehat{y_d},\ a_j^k and\ o_j^k$ for each node j in layer k by proceeding from layer 0, the input layer, to layer m, the output layer.
2) Backward phase: for each input-output pair $(\overrightarrow{x_d}, y_d)$, it must be calculated the results $\frac{\partial E_d}{\partial w_{ji}^k}$ for each weight $w_{ji}^k$ connecting the node 1 in layer k-1 to node j by proceeding from layer m, the output layer, to layer 1, the input layer.
    - Evaluate the error term for the final layer $\delta_1^m$.

- Backpropagate the error terms for the hidden layers $\delta_j^k$, working backwards from the final hidden layer k = m-1.
- Evaluate the partial derivatives of the individual error $E_d$ with respect to $w_{ij}^k$.

3) Combinations of gradients: for each input-output pair $\frac{\partial E_d}{\partial w_{ji}^k}$ the gradients must be combined to get the total gradient $\frac{\partial E(X,\theta)}{\partial w_{ji}^k}$ for the entire set of input-output pairs $X = \{(\overrightarrow{x_1}, \overrightarrow{y_1}), \ldots, (\overrightarrow{x_N}, \overrightarrow{y_N})\}$.
4) Weights updating: according to the learning rate $\alpha$ and total gradient $\frac{\partial E(X,\theta)}{\partial w_{ji}^k}$ it is possible to update the weights in the neural network.

There are two types of backpropagation Network: the static and recurrent back-propagation. The static is one kind of backpropagation network which produces a mapping of a static input for static output. It is useful to solve static classification issues like optical character recognition.

Recurrent backpropagation is fed forward until a fixed value is achieved. After that, the error is computed and propagated backward.

The main difference between both of these methods is: that the mapping is rapid in static back-propagation while it is non-static in recurrent backpropagation.

Given a multilayer neural network with activation f, the backpropagation algorithm can be summarised in these requirements:

- The training set
- The learning rate $\alpha$
- The optimization function that defines the error E
- A termination condition, which can be a maximum number of steps or a minimum error reduction rate.

The Dataset consists in input-output pairs $(\overrightarrow{x_i}, \overrightarrow{y_i})$ where $\overrightarrow{x_i}$ is the input and $\overrightarrow{y_i}$ is the desired output of the network on input $\overrightarrow{x_i}$.

The set of input-output pairs of size N is denoted

$$X = \{(\overrightarrow{x_1}, \overrightarrow{y_1}), \ldots, (\overrightarrow{x_N}, \overrightarrow{y_N})\}$$

In backpropagation, the parameters of primary interest are $w_{ij}^{k}$, the weight between node j in layer $l_k$ and node i in layer $l_{k-1}$, and $b_i^k$, the bias for node i in layer $l_k$. There are no connections between nodes in the same layer and layers are fully connected.

An error function, E(X,θ), which defines the error between the desired output $\overrightarrow{y_1}$ and the calculated output $\widehat{\overrightarrow{y_1}}$ of the neural network on input $\overrightarrow{x_1}$ for a set of input-output pairs $(\overrightarrow{x_\imath}, \overrightarrow{y_\imath})$∈X and a particular value of the parameters θ.

The generic approach to minimizing the error is by gradient descent, called back-propagation in this setting. Because of the compositional form of the model, the gradient can be easily derived using the chain rule for differentiation.

Training a neural network with gradient descent requires the calculation of the gradient of the error function E (X, θ) with respect to the weights $w_{ij}^{k}$ and biases $b_i^k$.

Then, according to the learning rate η, each iteration of gradient descent updates the weights and biases (collectively denoted θ) according to

$$\theta^{t+1} = \theta^{t} - \alpha \frac{\partial E(X, \theta^{t})}{\partial \theta}$$

Where $\theta^{t}$ denotes the parameters of the neural network at iteration t in gradient descent.

The derivation of the backpropagation algorithm is fairly straightforward. It follows from the use of the chain rule and product rule in differential calculus. Application of these rules is dependent on the differentiation of the activation function, one of the reasons the heaviside step function is not used (being discontinuous and thus, non-differentiable).

Assuming that the bias $b^k_i$ for node I in layer k is incorporated into the weights as $w^k_{0i}$ with fixed output of $o^{k-1}_0=1$ for node 0 in layer k-1. Therefore,

$$w_{0i}^{k} = b_i^{k}$$

The original formulation is the left part of this equation, while the right is the simpler one:

$$a_i^k = b_i^k + \sum_{j=1}^{r_k - 1} w_{ji}^k o_j^{k-1} = \sum_{j=0}^{r_k - 1} w_{ji}^k o_j^{k-1}$$

Backpropagation attempts to minimize the following error function with respect to the neural network's weights:

$$E(X,\theta) = \frac{1}{2N}\sum_{i=1}^{N}(\widehat{y_i} - y_i)^2$$

This would be possible by calculating for each weight the value of the error’s derivative.

Since the error function can be decomposed into a sum over individual error terms for each individual input-output pair, the derivative can be calculated with respect to each input-output pair individually and then combined at the end (since the derivative of a sum of functions is the sum of the derivatives of each function):

$$\frac{\partial E(X,\theta)}{\partial w_{ji}^k} = \frac{1}{N}\sum_{d=1}^{N}\frac{\partial}{\partial w_{ji}^k}\left(\frac{1}{2}(\widehat{y_i} - y_i)^2\right) = \frac{1}{N}\sum_{d=1}^{N}\frac{\partial E_d}{\partial w_{ji}^k}$$

In fact, Thus, for the purposes of derivation, the backpropagation algorithm will concern itself with only one input-output pair. Once this is derived, the general form for all input-output pairs in X can be generated by combining the individual gradients. Thus, the error function in question for derivation is:

$$E = \frac{1}{2}(\widehat{y_i} - y_i)^2$$

The goal is to minimise the error so to study the partial derivative of the error function.

$$\frac{\partial E}{\partial w_{ji}^k} = \frac{\partial E}{\partial a_j^k}\frac{\partial a_j^k}{\partial w_{ji}^k}$$

where $a_j^k k$ is the activation (product-sum plus bias) of node j in layer k before it is passed to the nonlinear activation function (in this case, the sigmoid function) to generate the output. This decomposition of the partial derivative basically says that the change in the error function due to a weight is a product of the change in the error function E due to the activation $a_j^k$ times the change in the activation $a_j^k$ due to the weight $w_{ji}^k$.

The definition of error is:

$$\delta_j^k = \frac{\partial E}{\partial a_j^k}$$

The second term can be calculated by the previous equation so the derivative of the error with respect of the weights is:

$$\frac{\partial E}{\partial w_{ji}^k} = \delta_j^k o_i^{k-1}$$

Therefore, the partial derivative of a weight is a product of the error term $\delta_j^k$ at node j in layer k, and the output $o_i^{k-1}$of node i in layer k-1. This makes intuitive sense since the weight $w_{ji}^k$ connects the output of node i in layer k-1 to the input of node j in layer k in the computation graph.

Note that these partial derivatives don't depend on a particular error function or activation function.

The most common error $\delta_j^k$ function is the mean squared error and the calculation of the error proceed from the output to the input, that is why this algorithm is called back-propagation or backwards propagations of errors.

The phase in which the neural network calculates the output precedes the backward phase for every iteration of gradient descent. In the forward phase, activations $a_j^k$ and outputs $o_j^k$ will be remembered for use in the backwards phase. Once the backwards phase is completed and the partial derivatives are known, the weights (and associated biases $b_j^k = w_{oj}^k$) can be updated by gradient descent.

This process is repeated until a local minimum is found or convergence criterion is met.

For the final layer, the error is:

$$\delta_1^m = g_o'(a_1^m)(\widehat{y_d} - y_d)$$

For the hidden layers the error is:

$$\delta_j^k = g'\left(a_j^k\right)\sum_{l=1}^{r^{k+1}} w_{jl}^{k+1}\delta_l^{k+1}$$

Combining the errors derivatives for each input-output pair it is possible to write the formula for updating the weights such as:

$$\Delta w_{ij}^k = -\alpha\frac{\partial E(X,\theta)}{\partial w_{ji}^k} = -\alpha\frac{1}{N}\sum_{d=1}^{N}\frac{\partial E_d}{\partial w_{ji}^k}$$

# 3. Traction separation law

Traction separations laws are specific constitutive laws in the theory of the cohesive zone model.

Cohesive zone models are based on cohesive interactions that approximate nonlinear fracture behaviour. These models are used to study stress singularities in linear elastic fracture mechanics and to approximate nonlinear material separation phenomena.

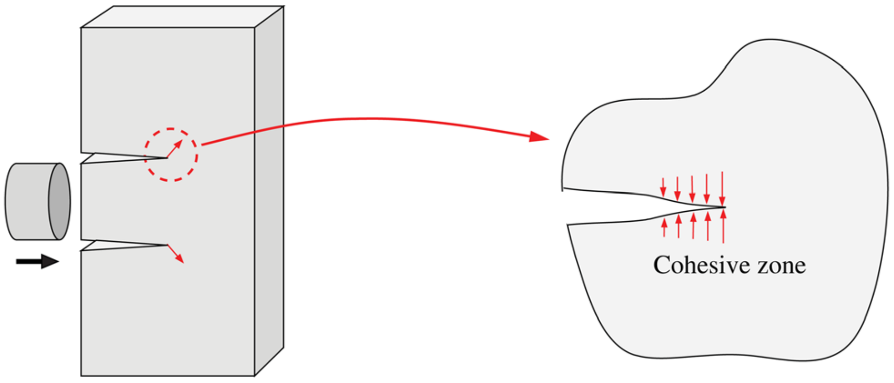


*Figure 3.1 Zoom of the cohesive zone*

The Hillerborg model, also known as Cohesive zone model (Fig. 3.1), assumes that the stress displacement behaviour (σ-δ ) observed in the damage zone of a tensile specimen is a material property.

Fracture formation is studied as a gradual phenomenon in which separation of the surfaces involved in the crack takes place across an extended crack tip, or cohesive zone, and is resisted by cohesive tractions.

The Cohesive Zone Model does not represent any physical material but describes the cohesive forces which occur when material elements are being pulled apart.

Cohesive interactions are generally a function of displacement separation and approximate progressive nonlinear fracture behavior. If the displacement jump is greater than a characteristic length ($\delta_n$), complete failure occurs because it doesn't have more load-bearing capacity.

Figure 3.2 (a) shows a schematic stress-displacement curve, and 3.2 (b) illustrates the idealization of the damage zone ahead of a growing crack. As the surfaces (known as cohesive surfaces) separate, traction first increases until a maximum is reached, and then subsequently reduces to zero which results in complete separation.

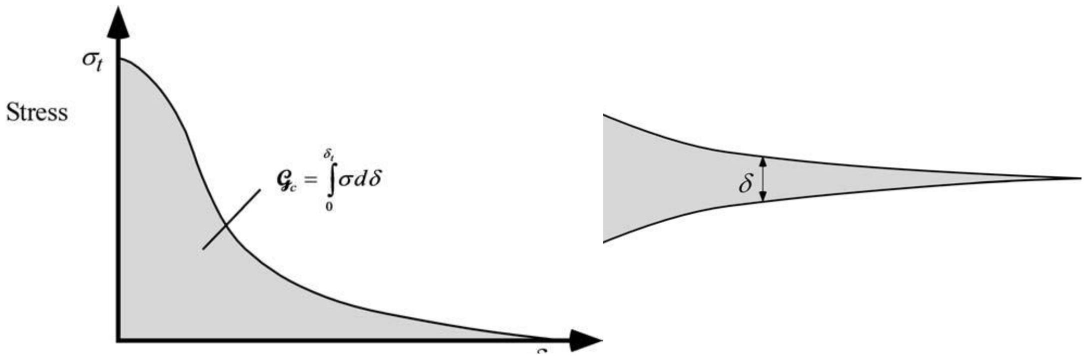


*Figure 3.2 (a) Stress-displacement response and (b) Damage zone ahead a crack.*

Cohesive zone model describes also crack nucleation and pervasive cracking through various time and length.

One of the fundamental aspects in cohesive zone modeling is the definition of the traction-separation relationship across fracture surfaces, which approximates the nonlinear fracture process.

In general, the initiation and continuation of crack growth depends on several factors, such as bulk material properties, body geometry, crack geometry, loading distribution, loading rate, load magnitude, environmental conditions, time effects (such as viscoelasticity or viscoplasticity), and microstructure.

There are three different ways of applying a force that can create a crack as shown in Fig 3.3:

- Mode I fracture: It is a opening mode where a tensile stress normal to the plane of the crack is applied;
- Mode II fracture: A in-plane shear stress is applied so It’s considered a sliding mode;
- Mode III fracture: It’s a tearing mode where a shear stress is applied parallel to the crack front and also to the plane of the crack ( Out-of-plane shear).

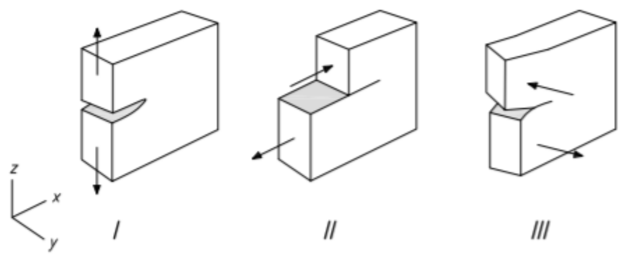


*Figure 3.3 Mode I, II, II as different ways to apply a force that lead to a crack*

Three families of traction-separation laws can be identified:

1. Elastic (reversible) models in which the t-w law derives from a potential function and as a consequence cannot correctly describe unloading processes.
2. Elasto-plastic models in which the formulation parallels that of classical continuum elastoplasticity , but vectorial relationship between tractions and separations are ser instead of stress and strain; since the stiffness of the crack just after initiation is nominally infinite, in the limit its behavior should be rigid-plastic, and these models are characterized by a very stiff unloading response.
3. Damage-based models obtained by exporting classical continuum damage approaches to vectorial, rather than tensorial, relationships, characterized by displaying linear unloading to the origin.

Cohesive traction-separation relationships can be studied as nonpotential-based models or potential-based models.

Nonpotential models are easy to develop because they don't require symmetry but they don't consider all separation paths. Potential-based models depend on the potential function, which is a characteristic of the fracture behavior and it is related to the fact that for close processes the work is non-negative.

The traction is in fact the first derivative of the fracture energy potential $(\Psi)$ and is considered as cohesive interaction over fracture surfaces. The second derivative of the energy potential is the constitutive relationship ( material tangent modulus).

The main problem with potential-based models is that they display limitations, especially for mixed-mode problems, because of the boundary conditions associated with cohesive fracture.

Cohesive traction-separation relationships may be obtained by employing theoretical, experimental and computational techniques.

The hypothesis of the cohesive constitutive laws can be summarized in:

- The Traction-separation law is independent from any body’s rigid motion.
- The work needed to create a new surface has a finite value of the fracture energy $\Gamma_0$.
- The mode I fracture energy is different from mode II
- There is a characteristic length, beyond that there is the failure, so the material has no more bearing capacity.
- Traction in the fracture surface generally decreases to zero, while separation increases under conditions softening, which determines a negative stiffness.
- It may exists a potential for cohesive constitutive laws and the energy dissipation during the unloading and reloading phases is independent of a potential.

## 3.1. Displacement based models

The traction-separation law (TSL) contains two parameters, the maximum traction sustainable by the element $T_0$ , and a maximum opening, the separation $\delta_0$, at which the element totally fails.

This law describes the relationship between the actual traction T and the separation distance $\delta$ as a function $T(\delta)$. This function can have different shapes depending on the material properties they consider.

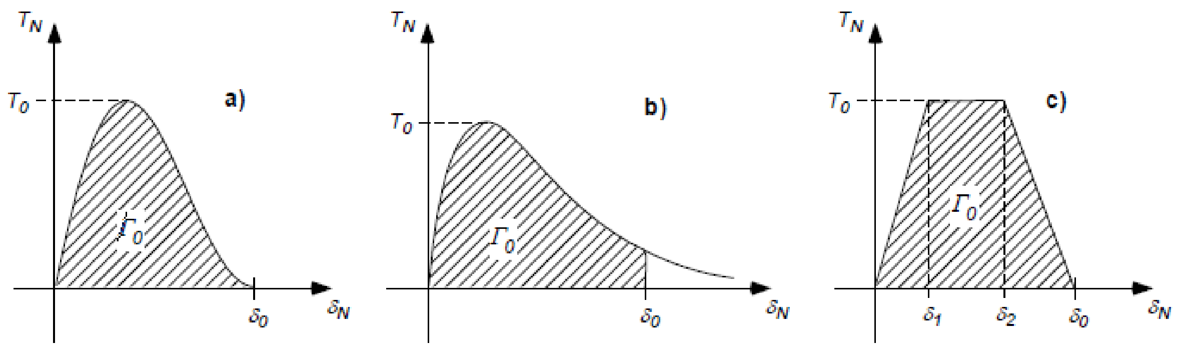


*Figure 3.4 different types of curves such as(a) cubic polynomial function, (b) exponential function and (c) trilinear law.*

Beside the two parameters $T_0$ and $\delta_0$, a third quantity can be defined, that is the energy dissipated by the cohesive element at total failure $\Gamma_0$.

It is calculated by the integral of the traction separation law and It is equal to the area subtended to the curve.

$$\Gamma_0 = \int_0^{\delta_0} T(\delta)d\delta$$

Effective displacements models are used for monotonic function so that fracture energy is constant regardless of the fracture mode. They cannot demonstrate the differences between positive separations and the negative ones. In fact, when the separation is growing, also the cohesive traction increases.

## 3.1.1. One-dimensional models

Tvergaard developed a displacement-based model in which he introduced the quantities $\overline{T}, \overline{\Delta}$ in relation with the normal and tangential component of the tractions as:

$$T_n = \frac{\overline{T}(\overline{\Delta})}{\overline{\Delta}}\frac{\Delta_n}{\delta_n}, \quad T_t = \frac{\overline{T}(\overline{\Delta})}{\overline{\Delta}}\alpha_e\frac{\Delta_t}{\delta_t}$$

Where $\delta_n$ *and* $\delta_t$ are respectively the normal ad tangent characteristic lengths in relation with the fracture energy; $\alpha_e$ is a nondimensional constant related with mode-mixity.

It is possible to describe an effective displacement which is nondimensional in relation with $\Delta_n and\ \Delta_t$,the normal and tangential separations described as

$$\bar{\Delta} = \sqrt{\left(\frac{\Delta_n}{\delta_n}\right)^2 + \left(\frac{\Delta_t}{\delta_t}\right)^2}$$

Tvergaard used a cubic polynomial function as in Fig. 3.4 (a) for the effective traction $\overline{\mathrm{T}}$ that describes the shape of the traction-separation relation.

$$\bar{T}(\bar{\Delta}) = \frac{27}{4}\sigma_{max}\,\bar{\Delta}(1 - 2\bar{\Delta} + \bar{\Delta}^2)$$

This equation corresponds to the normal cohesive traction proposed by Needleman.

In mode I, where $\Delta_t = 0$, the normal cohesive traction will be $T_n = \bar{T}(\bar{\Delta})$ and for mode II in which $\Delta_n = 0$, the tangent component of the traction will be $T_t = \alpha_e \bar{T}(\bar{\Delta})$.

In conclusion we can see that the constant $\alpha_e$ si a scaling factor between the two components of the cohesive traction.

There are other traction-separation relationships based on different kind of functions. Tvergaard and Hutchinson develop a trapezoidal function based on potential, in which they changed the definition of the $\alpha_e$ parameter as

$$\alpha_e = \frac{\delta_n}{\delta_t}$$

They defined the potential in relation to the displacement:

$$\Psi = \delta_n \int_0^{\bar{\Delta}} \bar{T}(\bar{\delta})\, d\bar{\delta}$$

The cohesive traction vector is the first derivative of the potential so it is possible to express it as:

$$T_n = \frac{\partial\Psi}{\partial\bar{\Delta}}\frac{\partial\bar{\Delta}}{\partial\Delta_n} = \frac{\bar{T}(\bar{\Delta})}{\bar{\Delta}}\frac{\Delta_n}{\delta_n}, \quad T_t = \frac{\partial\Psi}{\partial\bar{\Delta}}\frac{\partial\bar{\Delta}}{\partial\Delta_t} = \frac{\bar{T}(\bar{\Delta})}{\bar{\Delta}}\frac{\delta_n}{\delta_t}\frac{\Delta_t}{\delta_t}$$

This will result as follow:

$$\frac{\partial T_n}{\partial \Delta_t} = \frac{\partial T_t}{\partial \Delta_n}$$

This one dimensional potential leads to a symmetric system with an exact differential but it cannot distinguish different fracture energies along the normal and tangential directions.

Rose et al[1]. developed a one-dimensional energy given as:

$$\Psi = \delta_n \int_0^{\bar{\Delta}} e\sigma_{max}\bar{\delta}e^{\bar{\delta}}\, d\bar{\delta} = e\sigma_{max}\delta_n\left[1-(1+\bar{\Delta})e^{-\bar{\Delta}}\right]$$

Ortiz and Pandolfi develop another traction-separation relation keeping the same definition of $\alpha_e$ as the Tvergaard and Hutchinson model. They used a linear function without an initial slope.

They defined the cohesive traction vector in relation with the free energy density per unit area.

$$T_n = \frac{\tilde{T}(\tilde{\Delta})}{\tilde{\Delta}}\Delta_n, \quad T_t = \frac{\tilde{T}(\tilde{\Delta})}{\tilde{\Delta}}\Delta_t\beta_e^2$$

These two are the components of the Traction vector:

$$\boldsymbol{T} = \frac{\tilde{T}(\tilde{\Delta})}{\tilde{\Delta}}(\Delta_n\mathbf{n}_n + \boldsymbol{\Delta}_t\beta_e^2)$$

Where $\mathbf{n}_n$ is a normal unit vector to a cohesive surface and $\boldsymbol{\Delta}_t$is the in-plane tangential separation vector which is equal to $\boldsymbol{\Delta}_t\mathbf{n}_t$where $\mathbf{n}_t$is the tangential unit displacement vector.

They introduced the non-dimensional constant: $\beta_e = \frac{\delta_n}{\delta_t}$

Furthermore, $\tilde{\Delta}$ is now a dimensional component of $\Delta_n and\ \Delta_t$and it is related to $\bar{\Delta}$ as follow:

[1] The model has been used to investigate crack propagation of C-300 steel, functionally graded materials, and asphalt concrete.

$$\bar{\Delta} = \frac{\tilde{\Delta}}{\delta_n}$$

Ortiz and Pandolfi wrote that the initial elastic slope in the function equation might be a severe restriction for time step explicit integration.

Instead Geubelle and Baylor[2] developed a linear function model with the initial slope as the linear softening model. They introduced and internal residual strength variable called $D_s$ which is related to the effected displacement as

$$D_s = \min(D_{min}, max(0, 1 - \bar{\Delta}))$$

The bilinear cohesive model divide traction in the normal and tangential components as shown in these two equations:

$$T_n = \sigma_{max} \frac{D_s}{1-D_s} \frac{\Delta_n}{\delta_n}, T_t = \tau_{max} \frac{D_s}{1-D_s} \frac{\Delta_t}{\delta_t}$$

$D_{min}$ is a critical value that indicates when the cohesive traction reaches the cohesive strength. In other words, it is related to the effective displacement because if $\bar{\Delta} < 1 - D_{min}$, then the cohesive traction increases following a line as the separation increases too. If $\bar{\Delta} > 1 - D_{min}$there is the softening condition in which the two components of the Traction are written differently as follow:

$$T_n = \sigma_{max} \frac{1-\bar{\Delta}}{\bar{\Delta}} \frac{\Delta_n}{\delta_n}, T_t = \tau_{max} \frac{1-\bar{\Delta}}{\bar{\Delta}} \frac{\Delta_t}{\delta_t}$$

Where it is possible to substitute $\alpha_e = \tau_{max}/\sigma_{max}$ that leads to the equation

$$\bar{T} = \sigma_{max}(1 - \bar{\Delta})$$

[2] They used this model for studying the failure of polycrystalline brittle materials and viscoelastic asphalt concrete.

## 3.1.2. Three dimension model

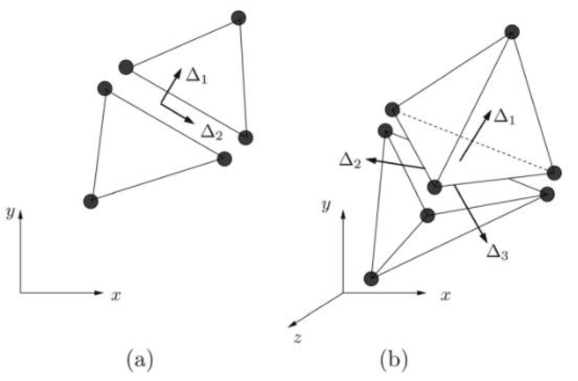


*Figure 3.5 local coordinate system (a)two-dimensions and (b) three-dimensions cohesive separations.*

It is possible to extend the previous equations to a 3D problem in which $\Delta_1$is an opening mode while $\Delta_2$and $\Delta_3$are two in-plane shear modes.

So it is possible to define the effective displacement as combination of the three dimensions:

$$\bar{\Delta} = \sqrt{\left(\frac{\Delta_1}{\delta_1}\right)^2 + \left(\frac{\Delta_2}{\delta_2}\right)^2 + \left(\frac{\Delta_3}{\delta_3}\right)^2}$$

Consequently, the traction components will be:

$$T_1 = \frac{\bar{\mathrm{T}}(\bar{\Delta})}{\bar{\Delta}} \frac{\Delta_1}{\delta_1}, T_2 = \frac{\bar{\mathrm{T}}(\bar{\Delta})}{\bar{\Delta}} \alpha_2 \frac{\Delta_2}{\delta_2}, T_3 = \frac{\bar{\mathrm{T}}(\bar{\Delta})}{\bar{\Delta}} \alpha_3 \frac{\Delta_3}{\delta_3}$$

Where $\alpha_2$ *and* $\alpha_3$ are the constant without dimension associated with the mode-mixity.

## 3.2. Potential-based models

Potential-based models have been demonstrated to have less limitations than the displacement-based models[3] because the potential is function of the two components of the separation vector instead of the effective displacement.

[3] As shown in 'Cohesive Zone Models: A Critical Review of Traction-Separation Relationships Across Fracture Surfaces' by Kyoungsoo and Park (2011).

The potential can be divided into a normal and tangential part. Each one of its components can be derived into their related traction component. Potential-based models are valid under the condition of monotonic separation paths. Thus, unloading/reloading relations should be addressed independently in order to describe energy dissipations, which include fatigue damage.

Needleman in 1987 developed for the traction separation function a cubic polynomial and a linear function to determine the tangential cohesive traction. He introduced a polynomial formula for a debonding potential, which is related to just the normal and tangential separations ($\Delta_n$ , $\Delta_t$) along the interface. Then, by deriving the potential, the interfacial normal and tangential tractions will be obtained.

$$\Psi(\Delta_n, \Delta_t) = \frac{27}{4}\sigma_{max}\delta_n \left\{\frac{1}{2}\left(\frac{\Delta_n}{\delta_n}\right)^2 \left[1 - \frac{4}{3}\left(\frac{\Delta_n}{\delta_n}\right) + \frac{1}{2}\left(\frac{\Delta_n}{\delta_n}\right)^2\right]\right.$$
$$\left. + \frac{1}{2}\alpha_s \left(\frac{\Delta_t}{\delta_n}\right)^2 \left[1 - 2\left(\frac{\Delta_n}{\delta_n}\right) + \left(\frac{\Delta_n}{\delta_n}\right)^2\right]\right\}$$

Where $\alpha_s$is the maximum traction carried by the interface under the mode I fracture condition, $\delta_n$is the characteristic length and $\sigma_{max}$ is the shear stiffness parameter.

It is possible to determine the tractions in the only case when $\Delta_n < \delta_n$ i.e. when the characteristic length is greater than the normal separation. In the other case where $\Delta_n > \delta_n$, there is not cohesive interactions.

$$T_n = \frac{\partial\Psi}{\partial\Delta_n} = \frac{27}{4}\sigma_{max}\left\{\left(\frac{\Delta_n}{\delta_n}\right)\left[1 - 2\left(\frac{\Delta_n}{\delta_n}\right) + \left(\frac{\Delta_n}{\delta_n}\right)^2\right] + \alpha_s\left(\frac{\Delta_t}{\delta_n}\right)^2\left[\left(\frac{\Delta_n}{\delta_n}\right) - 1\right]\right\}$$

$$T_t = \frac{\partial\Psi}{\partial\Delta_t} = \frac{27}{4}\sigma_{max}\left\{\alpha_s\left(\frac{\Delta_t}{\delta_n}\right)\left[1 - 2\left(\frac{\Delta_n}{\delta_n}\right) + \left(\frac{\Delta_n}{\delta_n}\right)^2\right]\right\}$$

The cohesive strength $\sigma_{max}$ is reached by $T_n$ when $\Delta_t = 0$ and $\Delta_n = \frac{\delta_n}{3}$.

This traction separation function is associated with the mode I fracture properties such as the cohesive strength and the fracture energy because the area under the curve when $\Delta_t = 0$ is the same as the fracture energy $\phi_n$.

Freed and Banks-Sills in 2008 started from Needleman's potential-based model with a cubic polynomial function. Their potential function is related to the mode mixity or phase angle $(\theta)$. The effective displacement and phase angle are:

$$\widetilde{\Delta} = \sqrt{\Delta_n^2 + \Delta_t^2}, \qquad \theta = \tan^{-1}\frac{\Delta_t}{\Delta_n}$$

The potential function is:

$$\Psi(\widetilde{\Delta}, \theta) = \frac{27}{4} t_0^*(\theta)\widetilde{\Delta}\left[\frac{1}{4}\left(\frac{\widetilde{\Delta}}{\delta_c^*(\theta)}\right)^3 - \frac{2}{3}\left(\frac{\widetilde{\Delta}}{\delta_c^*(\theta)}\right)^2 + \frac{1}{2}\left(\frac{\widetilde{\Delta}}{\delta_c^*(\theta)}\right)\right]$$

In this equation $t_0^*$ depends on the maximum cohesive strength and the phase angle as follow:

$$t_0^*(\theta) = \sigma_{max}\sqrt{1 + tan^2\,\theta}$$

$\delta_c^*$ depends also to the characteristic length:

$$\delta_c^*(\theta) = \delta_n\sqrt{1 + \tan^2\theta}$$

The last two definitions have meaning in the only case when $\widetilde{\Delta} < \delta_c^*(\theta)$ because if the effective displacement is greater than $\delta_c^*(\theta)$, then the cohesive tractions will be zero. It is also possible to write this potential equation in terms of normal and tangential separations.

In this way in mode I case, when the tangential separation is equal to zero, this potential is identical to the one from Needleman and in both their functions the tangential separations are quadratic.

The two tractions components will be derived not in term of effective displacement but regarding the separation components as follow, considering the case when $\widetilde{\Delta} < \delta_c^*(\theta)$:

$$T_n = \frac{\partial\Psi}{\partial\Delta_n} = \frac{27}{4}\sigma_{max}\left\{\left(\frac{\Delta_n}{\delta_n}\right)\left[\left(\frac{\Delta_n}{\delta_n}\right)^2 - 2\left(\frac{\Delta_n}{\delta_n}\right) + 1\right] + \frac{1}{2}\left(\frac{\Delta_t}{\delta_n}\right)^2\left[\left(\frac{\Delta_n}{\delta_n}\right) - \frac{4}{3}\right]\right\}$$

$$T_t = \frac{\partial\Psi}{\partial\Delta_t} = \frac{27}{4}\sigma_{max}\left\{\left(2\frac{\Delta_t}{\delta_n}\right)\left[\frac{1}{4}\left(\frac{\Delta_n}{\delta_n}\right)^2 - \frac{2}{3}\left(\frac{\Delta_n}{\delta_n}\right) + \frac{1}{2}\right]\right\}$$

Like Needleman's model, this one uses the cubic polynomials for the normal traction function and the tangential cohesive traction is linear.

There are other three models that are based on the concept of the universal binding energy developed by Rose et al.

They studied an atomistic potential that connects metallic binding energies and lattice parameters. The potential, called the universal binding energy, is defined as:

$$\Psi = -(1 + l)\exp(-l)$$

Note that this function is an exponential and l is the scaled separation.

After this definition, Rice and Wang obtained a new relationship for traction separation law for large tangential separation. Again, the Traction is derived from the potential in which $E_0$ is the initial modulus or one-dimensional tensile straining of the interface layer:

$$T_n(\Delta_n) = E_0\left(\frac{\Delta_n}{\delta_n}\right) axp\left(-\alpha_n \frac{\Delta_n}{\delta_n}\right)$$

Needleman developed another potential based on the universal binding energy maintaining the linear relationship for shear interaction as in his previous model.

$$\Psi(\Delta_n, \Delta_t) = \frac{9}{16}\sigma_{max}\delta_n\left\{1 - \left[1 + z\left(\frac{\Delta_n}{\delta_n}\right) - \frac{1}{2}\alpha_s\left(z\frac{\Delta_t}{\delta_n}\right)^2\right]\exp\left(-\frac{\mathrm{z}\Delta_n}{\delta_n}\right)\right\}$$

In fact, it is possible to obtain the tangential traction by deriving this potential and see that the traction is linear with respect to the tangential separation.

Then Needleman created an exponential-periodic potential, which is function of normal and tangential separations. The exponential-periodic potential is

$$\Psi(\Delta_n, \Delta_t) = \frac{\sigma_{max} e \delta_n}{z}\left\{1 - \left[1 + \left(\frac{z\Delta_n}{\delta_n}\right) - \beta_s z^2\left[1 - \cos\left(\frac{2\pi\Delta_t}{\delta_t}\right)\right]\right]\exp\left(-\frac{\mathrm{z}\Delta_n}{\delta_n}\right)\right\}$$

Where z=16e/9 and $\beta_s$ is a constant. The normal and tangential tractions are given by:

$$T_n = \frac{\partial\Psi}{\partial\Delta_n} = e\sigma_{max}\left\{\left(\frac{z\Delta_n}{\delta_n}\right) - \beta_s z^2\left[1 - \cos\left(\frac{2\pi\Delta_t}{\delta_t}\right)\right]\right\}\exp\left(-\frac{\mathrm{z}\Delta_n}{\delta_n}\right)$$

$$T_t = \frac{\partial \Psi}{\partial \Delta_t} = e\sigma_{max} \left\{ 2\pi\beta_s z \left(\frac{\delta_n}{\delta_t}\right) \sin\left(\frac{2\pi\Delta_t}{\delta_t}\right) \right\} \exp\left(-\frac{z\Delta_n}{\delta_n}\right)$$

In the condition where $\Delta_t = 0$ and $\Delta_n = \frac{\delta_n}{z}$ the normal cohesive strength $\sigma_{max}$ is reached. The normal traction has an exponential softening behavior, while the tangential traction illustrates periodic behavior. There fracture properties are valid just for mode I parameters like the fracture energy and the cohesive strength. This model based on exponential-periodic potential, doesn't comprehend the mode II so it is unable to describe general model based on mixed-mode fracture behaviours.

The generalization of this model was developed by Beltz and Rice who described the normal traction as an exponential function and the tangential traction was demonstrated by Peierls as a periodic function.

$$T_n = [B(\Delta_t)\Delta_n - C(\Delta_t)] \exp\left(\frac{-\Delta_n}{\delta_n}\right)$$

$$T_t = A(\Delta_n) \sin\left(\frac{2\pi\Delta_t}{\delta_t}\right)$$

Where $A(\Delta_n)$, $B(\Delta_t)$ and $C(\Delta_t)$ are function that satisfy the following boundary conditions.

There are two main notes about this model:

First of all the potential is an exact differential and second, the constant C (0) is equal to zero because the normal traction is zero when the two components of the displacements are zero.

So, in this case we have:

$$C(0) = 0$$

The area under a cohesive interaction represents the fracture energy so the normal traction of a cleavage fracture depends on the surface energy $\gamma_s$ and the tangential traction of a dislocation nucleation procedure is associated to the unstable stacking energy $\gamma_{us}$ as follow:

$$2\gamma_s = \int_0^\infty T_n(\Delta_n, 0)d\Delta_n = \phi_n$$

$$\gamma_{us} = \int_0^{\frac{\delta_t}{2}} T_t(0, \Delta_t)d\Delta_t = \phi_t$$

The two traction components satisfy the boundary condition when $\Delta_n = \infty$ that corresponds to the complete normal separation. This is why fracture surfaces cannot transfer tractions when there is complete separation along the normal direction.

In other words,

$$T_n(\infty, \Delta_t) = 0 \text{ and } T_t(\infty, \Delta_t) = 0$$

The generalized exponential-periodic potential of Beltz and Rice is:

$$\Psi = 2\gamma_s + 2\gamma_s \exp\left(\frac{-\Delta_n}{\delta_n}\right)\left\{\left[q + \left(\frac{q-r}{1-r}\right)\frac{\Delta_n}{\delta_n}\right]\sin^2\left(\frac{\pi\Delta_t}{\delta_t}\right) - \left[1 + \frac{\Delta_n}{\delta_n}\right]\right\}$$

Having the fracture energy and the cohesive strengths, it is possible to obtain the characteristic length parameters as follows:

$$\delta_n = \frac{\phi_n}{e\sigma_{max}}$$

$$\delta_t = \frac{\pi\phi_t}{\tau_{max}}$$

Another potential-based model was introduced by Xu and Needleman by changing the periodic function for tangential traction to an exponential expression. This is useful to characterize the interfacial shear failure. This model is called the exponential-exponential potential and it is shown as:

$$\begin{aligned}\Psi(\Delta_n, \Delta_t) = \phi_n& \\ &+ \phi_n \exp\left(-\frac{\Delta_n}{\delta_n}\right)\left\{\left[1 - r + \left(\frac{\Delta_n}{\delta_n}\right)\right]\frac{1-q}{r-1}\right. \\ &\left. - \left[q + \frac{r-q}{r-1}\frac{\Delta_n}{\delta_n}\right]\exp\left(-\frac{\Delta_t^2}{\delta_t^2}\right)\right\}\end{aligned}$$

The interfacial cohesive tractions as the potential's derivative are:

$$T_n = \frac{\phi_n}{\delta_n} \exp\left(-\frac{\Delta_n}{\delta_n}\right) \left\{ \frac{\Delta_n}{\delta_n} \exp\left(-\frac{\Delta_t^2}{\delta_t^2}\right) + \frac{1-q}{r-1} \left[1 - \exp\left(-\frac{\Delta_t^2}{\delta_t^2}\right)\right] \left[r - \frac{\Delta_n}{\delta_n}\right] \right\}$$

$$T_t = \frac{\phi_n}{\delta_n} \frac{2\delta_n}{\delta_t} \frac{\Delta_t}{\delta_t} \left[q + \frac{r-q}{r-1} \frac{\Delta_n}{\delta_n}\right] \exp\left(-\frac{\Delta_n}{\delta_n}\right) \exp\left(-\frac{\Delta_t^2}{\delta_t^2}\right)$$

Once again, as the previous model, it is possible to relate the fracture energies to the cohesive strength:

$$\phi_n = \sigma_{max} e \delta_n$$

$$\phi_t = \sqrt{e/2}\delta_t \tau_{max}$$

Where:

q is a constant and it is the ratio of the mode II fracture energy $\phi_t$ to the mode I fracture energy$\phi_n$, i.e.,

$$q = \frac{\phi_t}{\phi_n}$$

r is another nondimensional constant that depends on $\Delta_n^*$ which is the value of the normal separation when the normal traction is equal to zero.

$$r = \frac{\Delta_n^*}{\delta_n}$$

The condition when q=1 is reached when both the fracture energy of mode I and II are the same. In this case the effect of the r parameter disappears and the potential is simplified as

$$\Psi(\Delta_n, \Delta_t) = \phi_n - \phi_n \left[1 + \left(\frac{\Delta_n}{\delta_n}\right)\right] \exp\left(-\frac{\Delta_n}{\delta_n}\right) \exp\left(-\frac{\Delta_t^2}{\delta_t^2}\right)$$

The normal and tangential tractions not only demonstrate the exponentially decreasing softening but represent the different fracture parameters in mode I and II.

This exponential-exponential model has several limitations due to the introduction of parameters that are difficult to calculate as explained:

- The model contains this fracture parameter $\Delta_n^*$ that is difficult to obtain experimentally. In some cases, it is not a problem because it can disappear when q=1.
- This model in fact cannot be applied if fracture energy related to mode I is different to mode II fracture energy.
- It cannot control the elastic behaviour so it is difficult to do numerical simulations of cohesive surface elements.

The model doesn't correspond to reality when it reaches the final crack opening width because it would be infinite due to the exponential function.

**PPR, General Unified Potential-Based Model**

This general model was introduced to bypass the limitations of the exponential-exponential model so it is formulated with physical parameters and boundary conditions.

The parameters used are:

- Fracture energy
- Cohesive strength
- Shape
- Initial slope.

The boundary conditions that need to be satisfied by the potential-based model are as follow:

- Complete normal failure when$T_n(\delta_n, \Delta_t) = 0$ or $T_n(\Delta_n, \overline{\delta_t}) = 0$
- Complete tangential failure when $T_t(\Delta_n, \delta_t) = 0$ or $T_t(\overline{\delta_n}, \Delta_t) = 0$
- Mode I fracture energy $\int_0^{\delta_n} T_n(\Delta_n, 0) d\Delta_n = \phi_n$
- Mode II fracture energy $\int_0^{\delta_t} T_t(0, \Delta_t) d\Delta_t = \phi_t$
- Normal cohesive strength $T_n(\delta_{nc}, 0) = \sigma_{max}$ where $\frac{\partial T_n}{\partial \Delta_n}|_{\Delta_n = \delta_{nc}} = 0$
- Tangential cohesive strength $T_t(0, \delta_{tc}) = \tau_{max}$ where $\frac{\partial T_t}{\partial \Delta_t}|_{\Delta_t = \delta_{tc}} = 0$

As introduced above, in this model two shape parameters characterize various material softening responses. The potential of the PPR model defined as potential of mixed-mode cohesive fracture can be written as:

$$\Psi(\Delta_n, \Delta_t) = \min(\phi_n, \phi_t) + \left[\Gamma_n \left(1 - \frac{\Delta_n}{\delta_n}\right)^{\alpha} \left(\frac{m}{\alpha} + \frac{\Delta_n}{\delta_n}\right)^{m} + \langle \phi_n - \phi_t \rangle\right] X \left[\Gamma_n \left(1 - \frac{|\Delta_t|}{\delta_t}\right)^{\beta} \left(\frac{n}{\beta} + \frac{|\Delta_t|}{\delta_t}\right)^{n} + \langle \phi_n - \phi_t \rangle\right]$$

Where $\langle . \rangle$ is the Macaulary bracket defined as:

$$\langle x \rangle = \begin{cases} 0, (x < 0) \\ x, (x \geq 0) \end{cases}$$

The traction vector is obtained by deriving the potential. The potential's gradient is:

$$\mathrm{T_n}(\Delta_n, \Delta_t) = \frac{\Gamma_n}{\delta_n}\left[m \left(1 - \frac{\Delta_n}{\delta_n}\right)^{\alpha} \left(\frac{m}{\alpha} + \frac{\Delta_n}{\delta_n}\right)^{m-1} - \alpha \left(1 - \frac{\Delta_n}{\delta_n}\right)^{\alpha-1} \left(\frac{m}{\alpha} + \frac{\Delta_n}{\delta_n}\right)^{m}\right] X \left[\Gamma_t \left(1 - \frac{|\Delta_t|}{\delta_t}\right)^{\beta} \left(\frac{n}{\beta} + \frac{|\Delta_t|}{\delta_t}\right)^{n} + \langle \phi_n - \phi_t \rangle\right]$$

$$\mathrm{T_t}(\Delta_n, \Delta_t) = \frac{\Gamma_t}{\delta_t}\left[n \left(1 - \frac{|\Delta_t|}{\delta_t}\right)^{\beta} \left(\frac{n}{\beta} + \frac{|\Delta_t|}{\delta_t}\right)^{n-1} - \beta \left(1 - \frac{|\Delta_t|}{\delta_t}\right)^{\beta-1} \left(\frac{n}{\beta} + \frac{|\Delta_t|}{\delta_t}\right)^{\beta-1}\right] X \left[\Gamma_n \left(1 - \frac{\Delta_n}{\delta_n}\right)^{\alpha} \left(\frac{m}{\alpha} + \frac{\Delta_n}{\delta_n}\right)^{m} + \langle \phi_n - \phi_t \rangle\right] \frac{\Delta_t}{|\Delta_t|}$$

The normal and tangential tractions are defined within the cohesive interaction (softening) region where the fracture surface transfers cohesive normal and tangential tractions. All the characteristic parameters are the results of the boundary conditions.

The normal and tangential final crack opening widths ($\delta_n$, $\delta_t$) are the characteristic lengths and they are written as:

$$\delta_n = \frac{\phi_n}{\sigma_{max}} \alpha \lambda_n (1 - \lambda_n)^{\alpha-1} \left(\frac{\alpha}{m} + 1\right) \left(\frac{\alpha}{m} \lambda_n + 1\right)^{m-1}$$

$$\delta_t = \frac{\phi_t}{\tau_{max}} \beta \lambda_t (1 - \lambda_t)^{\beta-1} \left(\frac{\beta}{n} + 1\right) \left(\frac{\beta}{n} \lambda_t + 1\right)^{n-1}$$

The two energy, related to mode I and II are constants and when they are not equal they result as:

$$\Gamma_n = (-\phi_n)^{\frac{\langle \phi_n - \phi_t \rangle}{\phi_n - \phi_t}} \left(\frac{\alpha}{m}\right)^m$$

$$\Gamma_t = (-\phi_t)^{\frac{\langle \phi_t - \phi_n \rangle}{\phi_t - \phi_n}} \left(\frac{\beta}{n}\right)^n$$

Geometrically $\lambda_t$ and $\lambda_n$are the initial slope indicators and they are calculated as the ratio between the critical crack opening width and the final crack opening width. $\alpha \; and \; \beta$ are the shape parameters that provide choice when finding the right softening shape.

The constant exponents m and n are respectively:

$$m = \frac{\alpha(\alpha - 1)\lambda_n^2}{(1 - \alpha \lambda_n^2)} \; and \; n = \frac{\beta(\beta - 1)\lambda_t^2}{(1 - \beta \lambda_t^2)}$$

# 4. Granular material properties

Many geological materials, such as shale, mudstone, carbonate rock, limestone and rock salt are multi-porosity porous media in which pores of different scales may co-exist in the host matrix. When fractures propagate in these multi-porosity materials, these pores may enlarge and coalesce and therefore change the magnitude and the principal directions of the effective permeability tensors.

Soils are granular materials so their behaviour is determined by the forces between particles. These includes forces due to boundary loads (transmitted through the skeleton), particle forces such as gravitational and contact level forces such as capillarity.

Their static and dynamic behaviors are very complicated due to the complex interactions between particle and particle, particle and the liquid.

It is possible to understand the problem starting from a particle-scale because the granular material is discrete in nature rather than continuous.

Granular materials consist of grains in contact and surrounding voids. The micromechanical behaviour of granular materials is therefore inherently discontinuous and heterogeneous. In order to understand the mechanical behaviour of granular material from a microscopic point of view, it is important to understand the spatial distribution and orientation of grains and their contact conditions.

To really understand the mechanics of granular materials, particular interest goes to the particle rotation, contact moments as well as interparticle forces and contact displacement.

The intrinsic complexity of these materials can be divided into two different types of properties. Macroscopically, tensorial variables (stress tensor, strain tensor, fabric tensor) are commonly used based on Representative Volume Element (RVE), while vectors variables (contact force, contact displacement, contact normal) are adopted at particle-scale.

A tensor, called Fabric tensor, is introduced to characterize, in a tensor form, the spatial distribution of microscopic quantities such as particle orientation, contact normal and it can be used to derive constitutive equations.

## 4.1. Representative Volume Element

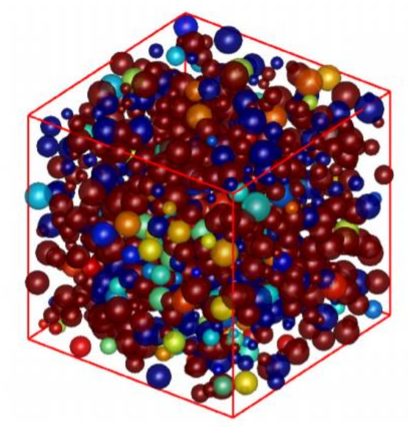


*Figure 4.1 Representative volume element for granular material*

Representative Volume Element (RVE) is a statistical representation of typical material properties.

RVE is defined by the representation of the material to be used to determine the corresponding effective properties for the homogenised macroscopic model with a size which is small enough compared to the macroscopic body and large enough compared to the microstructural size. An RVE should contain sufficient information about the microstructure and be a good representation of a continuum.

A typical RVE is composed by particles and voids among them. Three basics conditions are essential:

- RVE must be microscopically large enough, containing a sufficient number of particles and voids in order to get as many microscopic quantities as possible.
- It should be macroscopically small enough to be considered a spatial point.
- The characteristic length does not change over time and space.

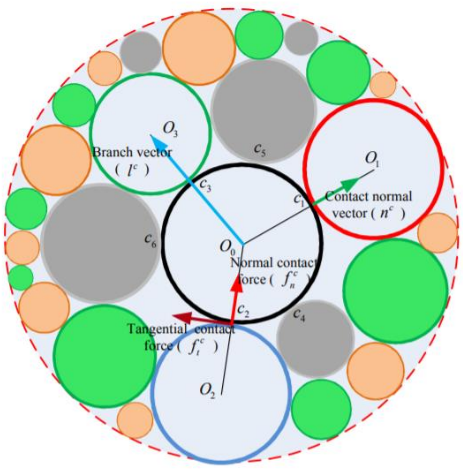
The choice of the RVE or its modelling determines the first difference between various homogenization theories. In particular, two classes of homogenization processes can be distinguished:

1. Homogenization methods for periodic media: The basic hypothesis in this case is that the medium may be described by a periodic number. The RVE is, in this case, the unit cell. With this approach, the treatment is completely deterministic. These models can account for precise local information such as the shape and orientations of inclusions.
2. Homogenization methods for media with randomly distributed phases: in this case it is not possible to give a deterministic description of the microstructure, so a statistical and probabilistic treatment becomes appropriate.

Considering a constitutive law t (δ,q) that predicts the traction vector t based on the history of the displacement jump δ over a cohesive-frictional surface with the normal direction vector being n.

The internal variables in q, if the cohesive surface is composed of a thin layer of granular materials, can be chosen among a large set of geometrical measures on micro-structural attributes such as porosity, coordination number and fabric tensor.

## 4.2. Microstructure Characterization



$O_3$
Branch vector ( $l^c$ )
$c_5$
$O_1$
$c_3$
$c_1$
Contact normal vector ( $n^c$ )
$c_6$
$O_0$
Normal contact force ( $f_n^c$ )
$c_2$
$c_4$
Tangential contact force ( $f_t^c$ )
$O_2$

*Figure 4.2 RVE and coordination number*

**Coordination number**

The coordination number is defined as the number of active contacts for each particle, where the normal contact force needs to be larger than zero. Taking the particle $O_0$ in the middle of a representative volume element in Fig. 4.2 for an example, six contacts including $c_1$, $c_2$, $c_3$, $c_4$, $c_5$ and $c_6$ can be found, which means its coordination number is 6.

For granular materials consisting of numerous particles, the averaged coordination number is usually adopted to characterize their connectivity and expressed as

$$C_n = \frac{N_{contact}}{N_{particle}}$$

Where $N_c$ is the number of particle contacts and $N_b$ is the number of the particles in the RVE. The coordination number is greater than zero.

## 4.3. Macroscopic Characterization

**Porosity**

Porosity Φ is a fraction of the total soil volume that is taken up by the pore space. It is the ratio between the void and the total volume of a representative element (RVE) of the material layer.

$$\Phi = \frac{V_{void}}{V_{total}}$$

Therefore, it is a single-value quantification of the amount of space available to fluid within a specific body of soil. It can range between 0 and 1, typically for soils is 0.3-0.7.

**Fabric tensor**

The fabric is a tensorial quantity which is used to characterize the internal structure of an assembly of grains. For a single particle its definition can be written as[4]:

[4] Stake (1982) defined fabric tensor for disc or spherical assemblies.

$$A_f = \frac{1}{N_{contact}} \sum_{c=1}^{N_{contact}} n^c \otimes n^c$$

Where $n^c$ is the contact normal unit vector of a particle contact c, c=1, 2,…, $N_{contact}$ in the RVE that is pointing outwards in the direction of the contact as $c_1O_1$ in Fig. 4.2.

Chantawarangul (1993) indicated that the fabric tensor can also be represented by appropriate distribution density function of contact normal:

$$A_f = \int_{\Omega} E(\Omega)\, n_i^c n_j^c \, d\Omega$$

Where $E(\Omega) = \frac{\left(1 + a_{ij}^r n_i^c n_j^c\right)}{4\,\pi}$ with $a_{ij}^r = a_{ji}^r\,, i \neq j.$

The principal values of tensors $a_{ij}^r$ namely $a_1^r, a_2^r\ and\ a_3^r$ are called coefficients of principal contact normal anisotropy or coefficients of contact anisotropy for brevity. These coefficients are related to the density of contact normal in principal contact directions. For an isotropic distribution of contacts, coefficients of contact anisotropy are zero and $E(\Omega) = \frac{1}{4\,\pi}$. A positive coefficient term implies a contact density in the corresponding principal direction which is greater than that expected of an isotropic assembly. Conversely, $a_i^r < 0$ implies that contact density is reduced below the density associated with an isotropic sample. The degree of fabric anisotropy can be represented by the second invariant $a_d^r$ written as:

$$a_d^r = \sqrt{\frac{3 a_{ij}^r a_{ij}^r}{2}}$$

The strong fabric tensor is:

$$A_{sf} = \frac{1}{N_{strongcontact}} \sum_{c=1}^{N_{strongcontact}} n^c \otimes n^c$$

Where $n^c$ is the normal unit vector of a strong particle contact (having a compressive normal force greater than mean contact force) c, c=1,2,…, N$_{\text{contact}}$ in the RVE.

**<u>Contact force and stress</u>**

Deformation of granular materials is accompanied by significant changes in the magnitudes of contact forces. The inter-particle forces can be decomposed into the direction normal and tangent to the contact planes, namely normal contact force $f_n^c$ and tangential contact force $f_t^c$, as shown in Fig. 4.2.

The normal contact force and tangential contact force can be expressed as:

$$F_{ij}^n = \frac{1}{4\pi}\int_\Omega \overline{f^n}\,(\Omega) n_i n_j d\Omega\,, \qquad F_{ij}^t = \frac{1}{4\pi}\int_\Omega \overline{f_i^t}\,(\Omega) n \quad n_j d\Omega$$

Where $\overline{f^n}$ and $\overline{f_i^t}$ are the density distribution function defined as:

$$\overline{f^n}(\Omega) = \overline{f_0^n}\left(1 + a_{ij}^n n_i n_j\right) \qquad \overline{f_i^t}(\Omega) = \overline{f_0^n}\left[a_{ij}^t n_j - (a_{kl}^t n_k n_l)\right]$$

Starting from the contact force definition, it is possible to express the stress tensor as:

$$\sigma_{ij} = \frac{1}{V}\sum_{c=1}^{N_c} f_j^c l_i^c$$

**<u>Contact displacement and strain</u>**

The contact displacements are generally characterized in terms of the translations of the particle centres, and the rigid-body rotations of the grains around their centre. It is possible to have displacement from macro-deformations of the RVE model in DEM simulations.

# 5. Data Generation

## 5.1.Discrete Element Method

The dataset used to develop the neural network was generated by a DEM simulation. The discrete element method proposed by Cundall and Strack (1979), also known as DEM is a very powerful numerical tool to simulate soils and other granular materials. The DEM modeling involves specifying the equations of motion for a system of discrete bodies and solving the resulting equations.

The mechanical response of granular materials in DEM is governed by the contacts between constitutive particles and between particles and the boundaries. So that the physical quantities that control these interactions (particle rotations, contact orientations, contact forces etc.) can easily be measured, which is almost impossible to capture in a laboratory test.

The DEM model gives a look at what is inside of the material and is capable to understand the fundamental particle interactions underlying the complex, macro-scale response. the DEM tries to solve large-displacement problems in geomechanics that are These problems cannot easily be modeled using more widespread continuum approaches such as the finite element method (FEM).

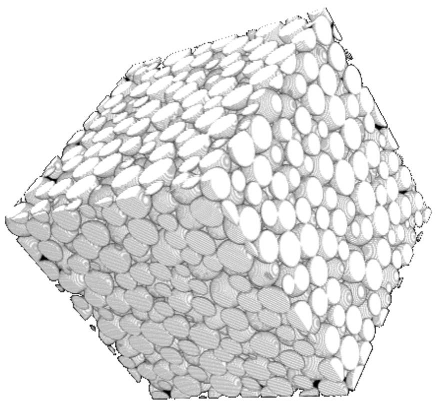


*Figure 5.1 Scheme of an RVE*

Two particles might establish a new interaction, which consists in:

1. Detecting collision between particles;
2. Creating new interaction and determining its properties (such as stiffness); they are either precomputed or derived from properties of both particles.

Then for already existing interactions, the following steps are:

- Strain evaluation;
- Stress computation based on strains;
- Force application to particles in interaction.

The DEM software YADE[5] is an open-source software, which is developed based on the C++ & Python programming languages. The calculation method is similar to the one proposed by Cundall and Strack (1979).

The contact laws governing the interactions between the particles are defined by the parameters illustrated in Figure 5.2.

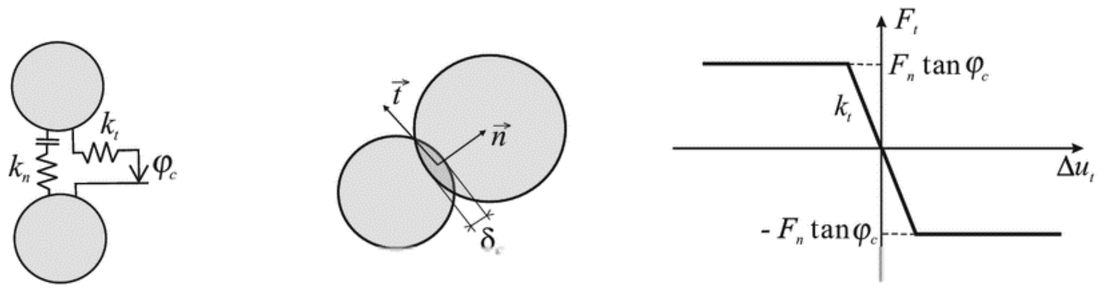


*Figure 5.2 Contact laws governing the interactions by Cundall and Strack (1979)*

The main parameters are the normal stiffness coefficient $k_n$ (normal direction to the contact plane), the tangential stiffness coefficient $k_t$ (tangent direction to the contact plane) and the microscopic friction angle $\varphi_c$. No tensile force is contemplated.

The inter-particle contact behaviour is governed by an elastic force-displacement relation in the normal contact direction as:

$$\Delta F_n = k_n \Delta\delta_c \ and \ F_n \geq 0$$

[5] YADE is the abbreviation of "Yet Another Dynamic Engine" by Šmilauer et al. (2010)

Where $\delta_c$ is the penetration depth between two particles in contact. The tangent force instead is calculated at each time step like:

$$\Delta F_t = k_t \Delta u_t \ and \ |F_t| \leq F_n \tan \varphi_c$$

For every time step, the tangential component of contact force must be corrected such that it does not exceed the shear strength of the contact. Then the resultant force and moment applied are updated with the two components of the contact force.

The phases of the DEM simulation are in loop as the following steps:

1. Update list of contacts
2. Calculate contact force
3. Calculate force and moments applying on each particle
4. Calculate the acceleration of each particle
5. Update the velocity and position of particles.

The numerical techniques used in DEM can be divided into two categories as soft sphere (molecular dynamics) and hard sphere (event driven) approaches. As in figure 5.3, the soft sphere model in which the particle is considered "soft", allows the penetration between particles because it considers the deformation at the contact point.

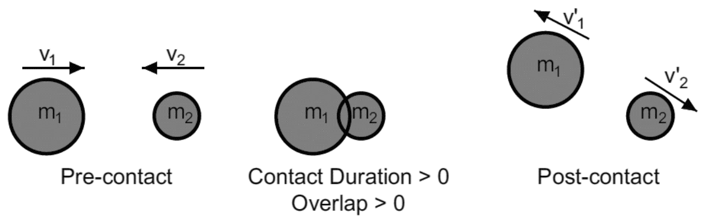


*Figure 5.3 Model of the soft sphere*

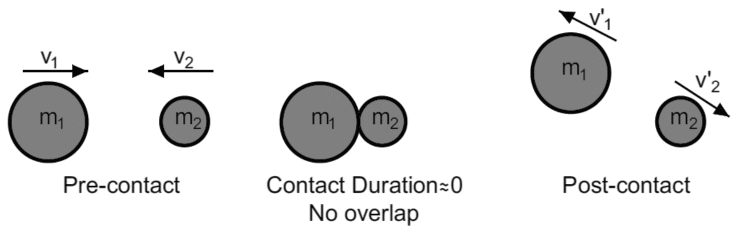


*Figure 5.4 Model of the hard sphere*

The principle behind the soft sphere method is to solve the equations governing the linear and angular dynamic equilibrium of contacting particles for every time step. In fact, the word soft may cause some misunderstanding; in the simulation, soft particles are actually rigid, however they are allowed to have overlap at the contact points. Consequently, physical actions are realized only when spheres enter each other.

In the YADE code, the soft sphere model is adopted so particle contact is allowed as seen in Fig. 5.5. When the model involves disk or sphere particles such as p and q, the contact overlap is calculated for a 3D model as:

$$\delta_c = R_p + R_q - \sqrt{\left(x_p - x_q\right)^2 + \left(y_p - y_q\right)^2 + \left(z_p - z_q\right)^2}$$

Where R is the radius of the single particle and x, y, z are the centroidal coordinates for each sphere.

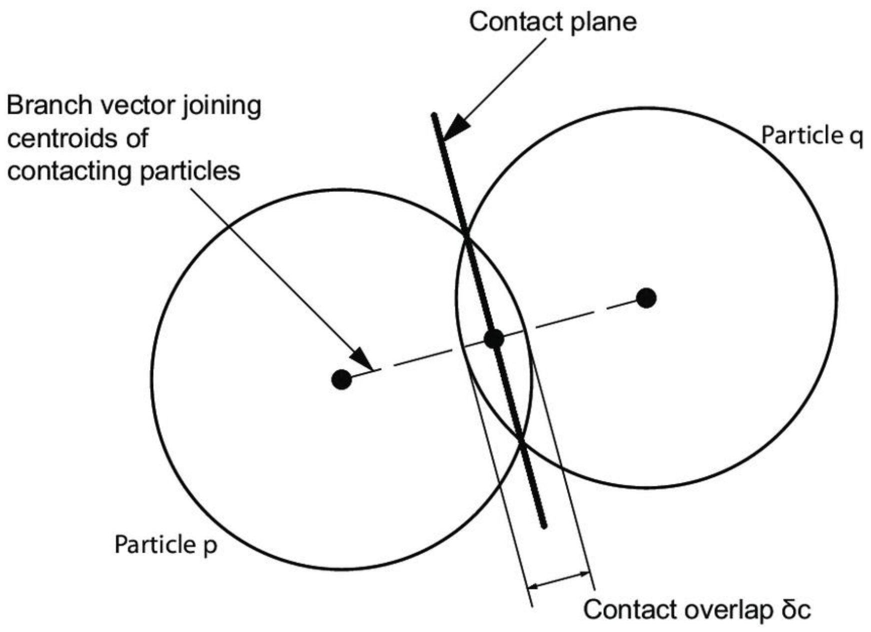


*Figure 5.5 Overlap between p and q sphere*

At the end, if the quantity $\delta_c$ is positive the force transmitted will be a compression, otherwise the contact will be classified as inactive.

**Creating interaction between particles**

The exact collision detection depends on the geometry of individual particles but in Yade terminology, the Collider creates only potential interactions.

It is possible to refer to kinematic variables of the contacts as ‘strains’, although at this scale it is also common to speak of ‘displacements’.

Basic DEM interaction defines two stiffnesses: normal stiffness $K_N$ and shear (tangent) stiffness $K_T$. It is desirable that $K_N$ be related to fictitious Young’s modulus of the particles’ material, while $K_T$ is typically determined as a given fraction of computed $K_N$. The $\frac{K_T}{K_N}$ ratio determines macroscopic Poisson’s ratio of the arrangement, which can be shown by dimensional analysis: elastic continuum has two parameters ($E$ and $v$) and basic DEM model also has 2 parameters with the same dimensions $K_N$ and $\frac{K_T}{K_N}$; macroscopic Poisson’s ratio is therefore determined solely by $\frac{K_T}{K_N}$ and macroscopic Young’s modulus is then proportional to $K_N$ and affected by $\frac{K_T}{K_N}$.

**Normal stiffness**

The algorithm commonly used in Yade computes normal interaction stiffness as stiffness of two springs in serial configuration with lengths equal to the sphere radii as seen in Fig. 5.2:

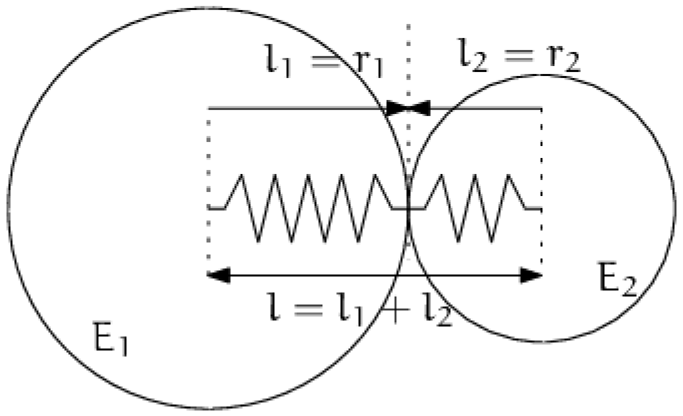


Figure 5.6 Series of 2 springs representing normal stiffness of contact

It is possible to define the distance $l = l_1 + l_2$ where $l_i$ are the distances between contact point and sphere centres that are initially equal to the sphere radius. Change of distance between the sphere centres $\Delta l$ is distributed onto deformations of both spheres $\Delta l = \Delta l_1 + \Delta l_2$ proportionally to their compliances. Displacement change $\Delta l_i$ generates force $F_i = \mathrm{K_i}\Delta l_i$, where $K_i$ assures proportionality and has physical meaning and dimension of stiffness; $K_i$ is related to the sphere material modulus $E_i$.

## 5.2. Generated data

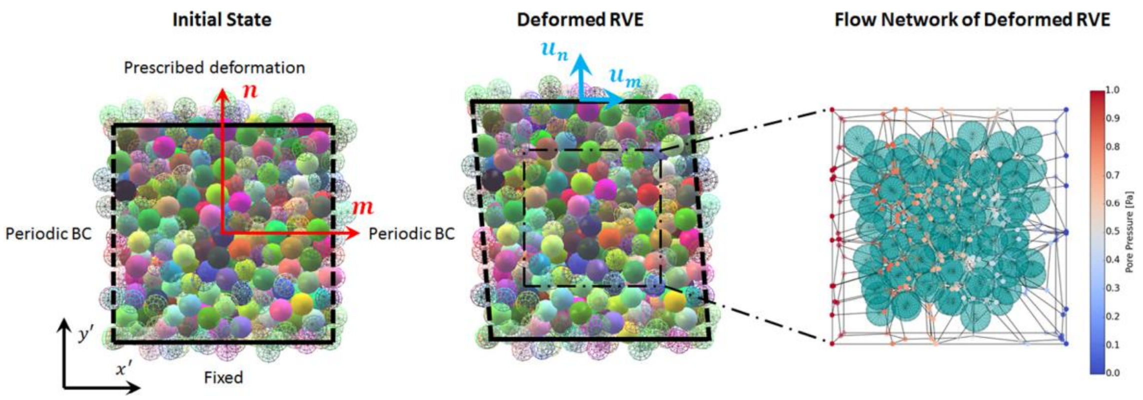


*Figure 5.7 Scheme of the RVE simulation*

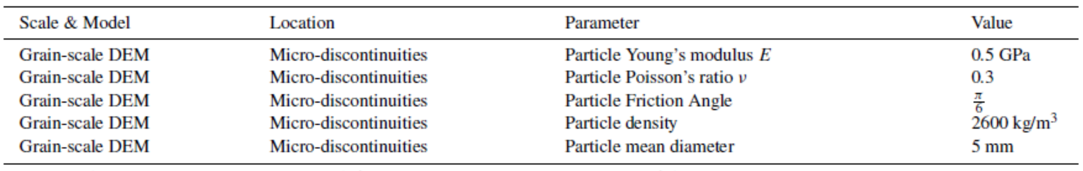
In each RVE simulation, the displacement boundary conditions are prescribed as shown in Fig. 5.7. The size of the DEM RVE is 10 cm x 10 cm x 5 cm, while the averaged grain diameter is 0.5 cm. A set of displacement jump paths {un, us} is applied to the microscale RVE, and the tractions {tn, ts} are homogenized at each incremental deformation step.

Before the displacement-driven grain-scale simulation begins, the DEM assembly must be in the stress state consistent to the macroscopic boundary condition. This is achieved by subjecting the DEM assembly with the right amount of shear and normal tractions along the boundaries.

Discrete Element Method (DEM) has been used to study the micro-mechanisms of granular materials. By considering their discrete nature, DEM calculates the interactions between each contact at every time-step. This is capable of telling how particles become arranged in space to form an internal structure.

In fact, we start from microstructure characterization in order to understand the macro-scale problem.

Table 5.1 Parameters chosen for the simulation



| Scale & Model | Location | Parameter | Value |
|---|---|---|---|
| Grain-scale DEM | Micro-discontinuities | Particle Young's modulus $E$ | 0.5 GPa |
| Grain-scale DEM | Micro-discontinuities | Particle Poisson's ratio $\nu$ | 0.3 |
| Grain-scale DEM | Micro-discontinuities | Particle Friction Angle | $\frac{\pi}{6}$ |
| Grain-scale DEM | Micro-discontinuities | Particle density | 2600 kg/m$^3$ |
| Grain-scale DEM | Micro-discontinuities | Particle mean diameter | 5 mm |

The parameters chosen for the simulation are:

Table 5.2 Properties of the particles

| *Parameter of the particle* | *Value* |
|---|---|
| *Young's Modulus E* | 0.5 GPa |
| *Poisson's ratio v* | 0.3 |
| *Friction angle* | 30° |
| *Density* | 2600 kg/m$^3$ |
| *Mean diameter* | 5 mm |

The gravel particle is assembled into a cube of 2000 spheres generated randomly. Then the program generates random loading cases changing the angle of the displacement applied to the RVE from 0° to 90°. In total the cases are 200.

The load cases consist in pairs of numbers that are the sin and cosine of the angle of the vector applied.

For each input parameters, the programs set up different assemblies that represent different connectivity between the particles. So, they will have different behaviours.

The material represented with this simulation is gravel and the forces applied are maximum 10 MPa. At the end the data as well as the microscopic properties will be collected in a csv file.

# 6. Program design

The steps to implement a neural network are described in the following paragraphs as functional parts of the code. Each phase has an infinite number of variables and ways to write it. The next subchapters focus only in the features developed in this thesis and do not comprehend all of the models that can possibly be implemented in a neural network.

## 6.1. Python setup

The code object of this thesis is written in Python language. Python is a high-level, general-purpose programming language created by Guido van Rossum and first released in 1991.

*Figure 6.1 Python logo*

Anaconda is a distribution of Python programming language for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

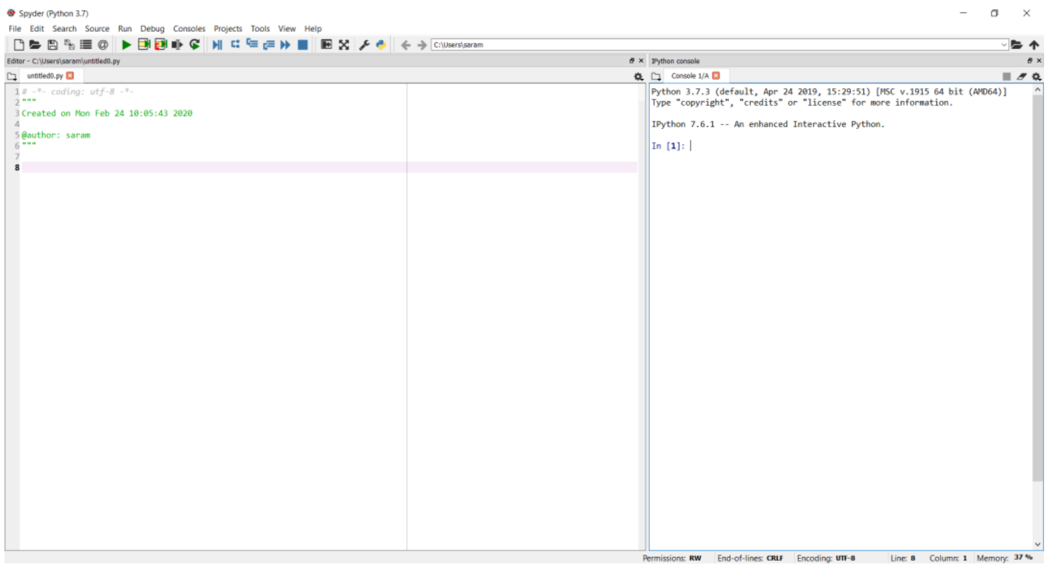
The version 3.7 of Python is installed in the Anaconda environment.

*Figure 6.2 Anaconda logo*

Anaconda navigator is a graphical user interface that allows to manage conda packages and launch applications. The code was written through the Spyder application.

Spyder interface software was installed in Anaconda. It allows to program easily in the Python language.

The Spyder interface includes an editor with an editor with syntax highlighting, introspection, code completion as seen in the Figure 6.3 below:


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\saram
Editor - C:\Users\saram\untitled0.py
untitled0.py
IPython console
Console 1/A

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 24 10:05:43 2020

@author: saram
"""

```

```
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.6.1 -- An enhanced Interactive Python.

In [1]:
```

```
Permissions: RW   End-of-lines: CRLF   Encoding: UTF-8   Line: 8   Column: 1   Memory: 37 %
```

*Figure 6.3 Python interface*

For the code object to this thesis, various libraries were integrated in the Spyder environment to help the scientific programming in Python.

Below a brief explanation of each library will be explained:

**TensorFlow**

This is an open-source library for dataflow and differentiable programming. It was developed by Google Brain for internal use in Google but then released in 2015.

With this library it is possible to create models for neural networks from training to testing a dataset and allows to import the Keras library.

**Keras**

Keras is an open-source library released in 2015. It is designed to enable fast experimentation with deep neural networks. In Keras, it is possible to assemble layers to build models. A model is usually a graph of layers and it helps to build a simple, fully-connected network such as a multi-layer perceptron.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing Deep Neural Network code. It supports other common utility layers like dropout, the activation function and batch normalization.

**NumPy**

This library allows to create arrays and calculate heavy operations with matrix and arrays.

**Pandas**

It is a software library for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. Pandas is mainly used for machine learning in form of dataframes. Pandas allow importing data of various file formats such as csv, excel etc.

**Scikit-learn**

It features various classification, regression and clustering algorithms including support vector machines.

**MatPlotlib**

This is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.

## 6.2. Load Data

The first step is to define the functions and classes used in this programme. In order to load the data set it is necessary to use the NumPy library.

The Dataset is created by a DEM simulation explained in the previous chapter and the data describes features of a granular material.

All of the input variables that describe each load case are numerical. This makes it easy to use directly with neural networks that expect numerical input and output values, and ideal a simple neural network in Keras. The problem with implementing other neural network is that usually it is necessary to start from pictures or data that are not necessarily numbers. So the problem is how to convert every input in mathematical terms.

The code will be learning a model to map rows of input variables (X) to an output variable (y), which we often summarize as $y = f(X)$.

The variables can be summarized as follows:

Input Variables (X): selected columns of the csv file.

Output Variables (y): selected columns of the csv file.

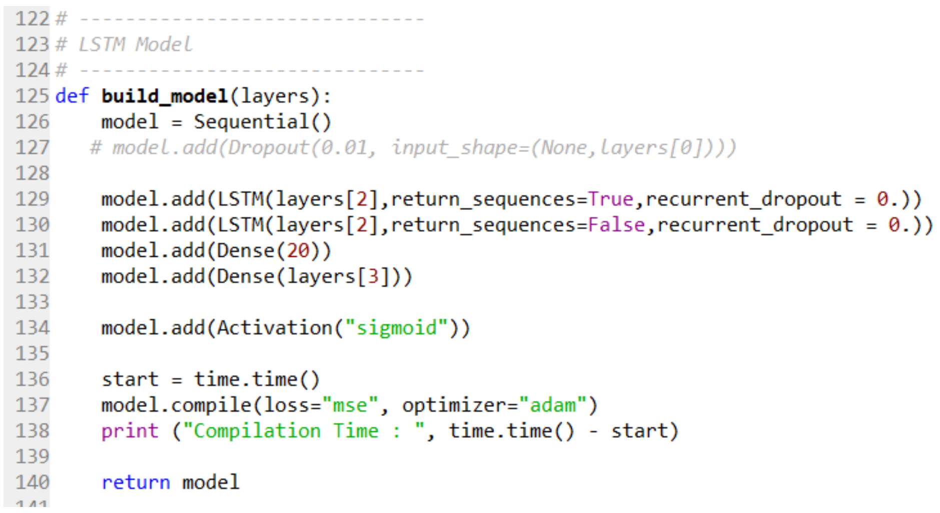
Once the CSV file is loaded into memory, it is possible to split the columns of data into input and output variables.

The data will be stored in a 2D array where the first dimension is rows and the second dimension are columns, e.g. [rows, columns].

## 6.3.Define Keras Model

The model created with keras is a Sequential model that defined by sequence of layers.

The first thing to do is to ensure the input layer has the right number of input features.



```
# -----------------------------
# LSTM Model
# -----------------------------
def build_model(layers):
    model = Sequential()
   # model.add(Dropout(0.01, input_shape=(None,layers[0])))

    model.add(LSTM(layers[2],return_sequences=True,recurrent_dropout = 0.))
    model.add(LSTM(layers[2],return_sequences=False,recurrent_dropout = 0.))
    model.add(Dense(20))
    model.add(Dense(layers[3]))

    model.add(Activation("sigmoid"))

    start = time.time()
    model.compile(loss="mse", optimizer="adam")
    print ("Compilation Time : ", time.time() - start)

    return model
```

*Figure 6.4 line of the code with implementation of layers*

Some of the features of the neural network have to be part of the trial and error process to find the best fit for the model.

Generally, the network must be large enough to capture the structure of the problem. Furthermore, it is not possible to calculate the most efficient number of layers or the number of nodes to use per layer in an artificial neural network to address a specific real-world predictive modeling problem.

The number of layers and the number of nodes in each layer are model hyperparameters that must be specified.

In this thesis three types of models will be developed as described in the following subchapters: The Dense, the LSTM and GRU algorithms.

### 6.3.1. Dense

The definition of the Dense algorithm is:

"*Dense* implements the operation: *output* = *activation(dot(input*, *kernel)* + *bias)* where *activation* is the element-wise activation function passed as the *activation* argument, *kernel* is a weights matrix created by the layer, and *bias* is a bias vector created by the layer (only applicable if *use_bias* is *True*).”[6]

It is a regular densely-connected neural network layer. The model dense is used for few and monotonic data where there is no history to train and predict such as curves of loading and unloading.

Therefore it is commonly used for elastic materials that are independent from time. In the code written, it is possible to see how bad the algorithm predicts the non-monotonic data.

The function has several arguments as follows in the model:

- Units: Positive integer, dimensionality of the output space. In this case the first to layers has 40 units and the third has just 2.
- Activation: activation function to use. In this case the sigmoid activation function is applied.
- Input_dim: positive integer, is the dimension of the input equal to 2.
- Dropout: it consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.

[6] From Keras documentation, https://keras.io/layers/core/

## 6.3.2. LSTM

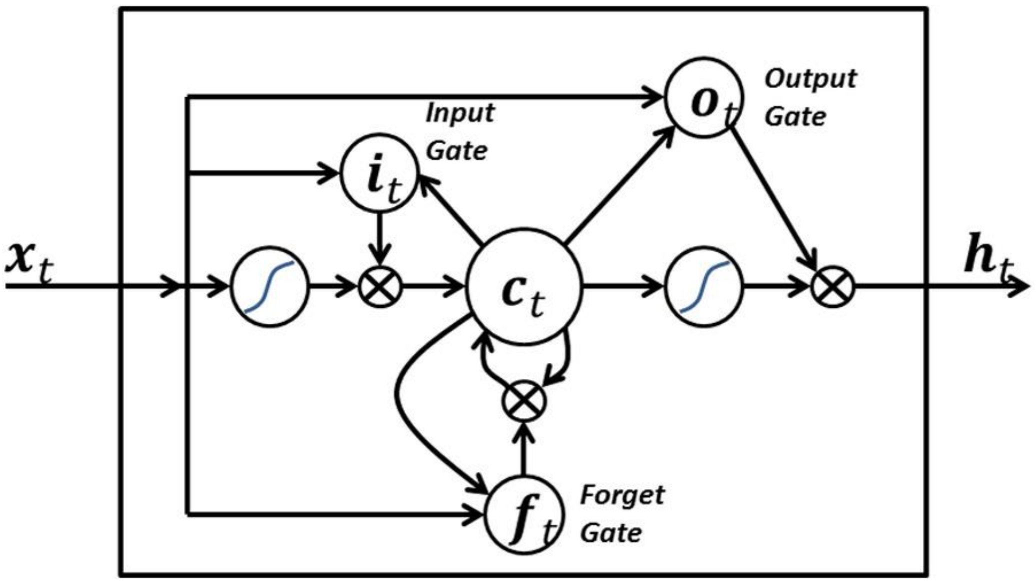


*Figure 6.5 A LSTM neuron with input, output and forget gate to process sequence with memory effect*

LSTM also known as Long Short-Term Memory, is a technique commonly used in computational linguistics. It was first introduced by Hochreiter and Schmidhuber in 1997 to create a neural network capable of having memory. LSTM uses memory blocks and a new entity called "gate" is introduced to control the flow of information and the state of the block as shown in Fig. 6.5

A LSTM neuron possesses a state of the memory cells at time t $C_t$. In this process there are some variables like $x_t$ that is the value of the input sequence at time t and $h_t$ is the value of the output sequence at time t.

The signal through the forget gate is given by

$$f_t = \sigma\left(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f\right)$$

Where $\sigma$ is the sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$, $W_f$ $and$ $U_f$ are weight matrices, $b_f$ the bias vector for the forget gate.

The new information to be stored in the cell state is given by the signal $i_t$ through the input gate

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i)$$

Where $W_i$ *and* $U_i$ are weight matrices and $b_i$ is the bias vector for the input gate.

The new candidate value cell state is given by a tanh layer:

$$\widetilde{C_t} = \tanh(W_C x_t + U_C h_{t-1} + b_C)$$

where tanh is the hyperbolic tangent function $\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$, $W_C$ *and* $U_C$ are weight matrices, $b_C$ is bias vector.

The old cell state $C_{t-1}$ is updated by the above forget and input information, i.e.,

$$C_t = f_t C_{t-1} + i_t \widetilde{C_t}$$

Finally, for the output signal

$$h_t = o_t \tanh(C_t)$$

where $o_t$ is the signal through the output gate

$$o_t = \sigma(W_0 x_t + U_o h_{t-1} + b_o)$$

where $W_o$ and $U_o$ are weight matrices, $b_o$ is bias vector for the output gate.

LSTM neural network accepts sequences of history values of the physical parameters as inputs.

The building and training of the LSTM data-driven model contains four steps. Firstly, the data of numerical simulations are stored in comma-separated values (CSV) file and are imported by an open-source Python data analysis library Pandas. The data are split into input features and outputs.

Each sequence of input and output is re-scaled to be within [0, 1] using the MinMaxScaler class in sklearn.preprocessing toolkit . The input data structure that can be processed by the LSTM model must be an array of dimension 3, where the entries for the first dimension are the samples, the second dimension are the time history steps and the last dimension are the input features.

After the neural network is built, the training parts of the epochs starts. It is possible feed the LSTM model with the preprocessed input and output data. The back-propagation algorithm will modify the weights of the neural network iteratively and the loss will be reduced to a small number (about $5 * 10^{-5}$ in this work). The learning rate can be reduced when the convergence becomes slow.

During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weight. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn’t contribute too much learning.

This type of algorithm as well as the less common GRU is used for a large amount of data- the csv file contains more than 16 thousand rows that correspond to the total amount of data. Because they have memory, they are able to learn and predict well plastic material in general such as the object of this thesis.

In opposite with the Dense layer, they have memory and take account of the timesteps regarding the input data.

### 6.3.3. GRU

The Gated Recurrent unit, also known as GRU, was introduced by Cho et al. in 2014 in order to solve the vanishing gradient problem. It can be considered as a standard recurrent neural network and a variation of the LSTM algorithm.

To solve the vanishing gradient problem of a standard RNN, GRU uses, so-called, update gate and reset gate. Basically, these are two vectors which decide what information should be passed to the output. The special thing about them is that they can be trained to keep information from long ago, without washing it through time or remove information which is irrelevant to the prediction.

In order to explain the mathematics behind the process, a single unit will be examined from the following recurrent neural network:

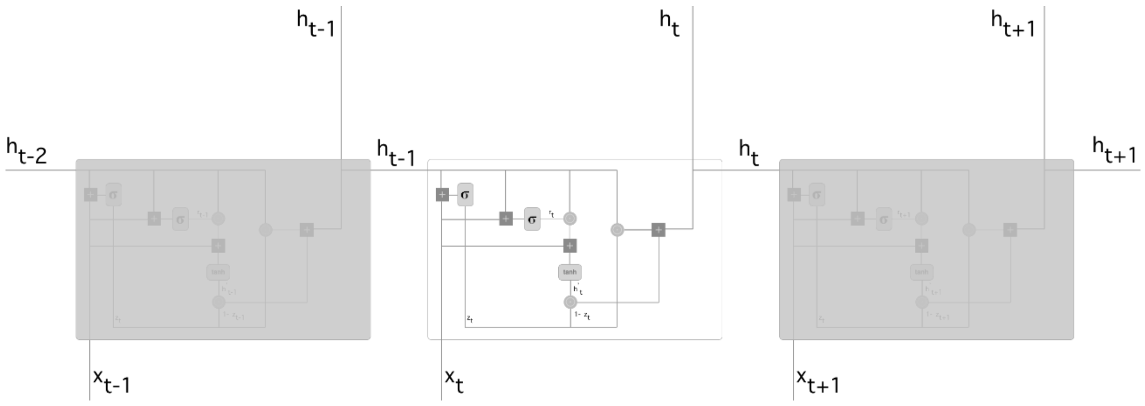


*Figure 6.6 neural network with Gated recurrent unit*

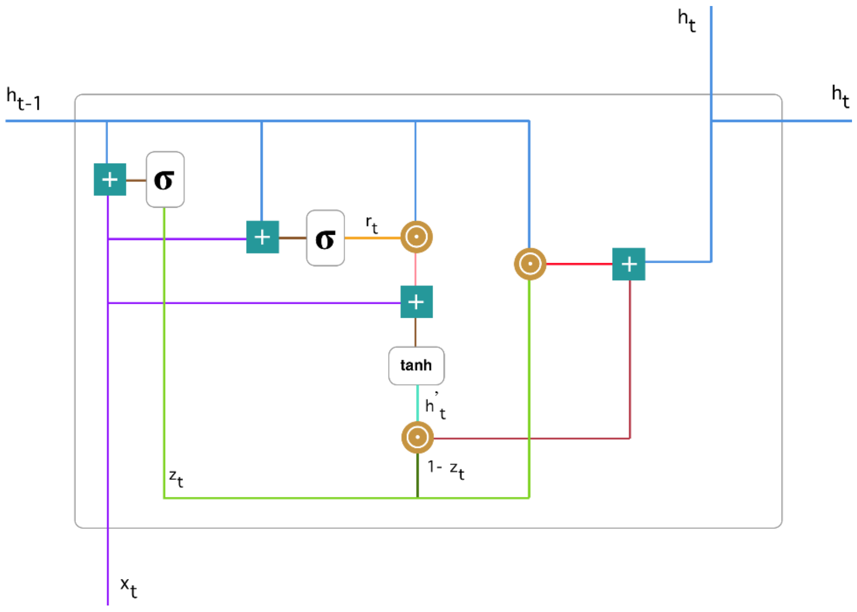


*Figure 6.7 Gated Recurrent Unit*

In Fig 6.7 it is possible to see different symbols that are used in this network such as:

| + | σ | ⊙ | tanh |
|---|---|---|---|
| "plus" operation | "sigmoid" function | "Hadamard product" operation | "tanh" function |

## Update Gate

The update gate helps the model to determine how much of the past information (from previous time steps) need to be passed along to the future.

It is the left part of Fig. 6.7, where the update gate $z_t$ is calculated for time step t using the formula:

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

When $x_t$ is plugged into the network unit, it is multiplied by its own weight $W^{(z)}$. The same goes for $h_{t-1}$which holds the information for the previous t-1 units and is multiplied by its own weight $U^{(z)}$. Both results are added together and a sigmoid activation function is applied to squash the result between 0 and 1.

**Reset gate**

Essentially, this gate is used from the model to decide how much of the past information to forget. It can be expressed as:

$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

This formula is the same as the one for the update gate. The difference comes in the weights and the gate's usage. As the previous step, looking and the next step of Fig. 6.7, the blue line represents $h_{t-1}$ while the purple line is $x_t$. These two parameters must be multiplied themselves for the corresponding weights, sum the results and apply the sigmoid function.

**Current memory content**

A new memory content is introduced that will use the reset gate to store the relevant information from the past. It is calculated as follows:

$$h'_t = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

The input $x_t$ is multiplied with his weight W and $h_{t-1}$ with U. Then the Hadamard (element-wise) product is calculated between the reset gate $r_t$ and $Uh_{t-1}$. That will be determine what to remove from the previous time steps. The neural network will learn to assign $r_t$ vector close to 0, forgetting about the previous time steps, focusing only on the last sentences.

Then the results of the two previous steps are summed up and the nonlinear activation function tanh is applied.

**Final memory at current time steps**

As the last step, the network needs to calculate vector $h_t$ which holds information for the current unit and passes it down to the network. This is done by the update gate that determines what to collect from the current memory content $h'_t$ and from the previous step $h_{t-1}$.

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t$$

The model can learn to set the vector $z_t$ close to 1 and keep a majority of the previous information. Since $z_t$ will be close to 1 at this time step, 1-$z_t$ will be close to 0 which will ignore big portion of the current content that is irrelevant for the prediction.

## 6.4. Compile Keras Model

Once that the model has been defined, it can be compiled. Compiling the model can be easy by using several efficient numerical libraries such as TensorFlow. The backend automatically chooses the best way to represent the network for training and making predictions to run on the hardware.

When compiling, some additional properties useful to better predict the data are required and must be specified. Training a network means finding the best set of weights to map inputs to outputs in our dataset. In fact, these parameters help the network to improve itself.

A loss function must be defined in order to evaluate a set of weights and also the optimizer, which is used to search through different weights for the network and any optional metrics that are likely to collect and report during training.

In this case, we will use mean squared error as the loss argument. The Mean Squared Error, or MSE, loss is the default loss to use for regression problems.

Mathematically, it is the preferred loss function under the inference framework of maximum likelihood if the distribution of the target variable is Gaussian.

Mean squared error is calculated as the average of the squared differences between the predicted and actual values. The result is always positive regardless of the sign of the predicted and actual values and a perfect value is 0.0. The squaring means that larger mistakes result in more error than smaller mistakes, meaning that the model is punished for making larger mistakes.

It will be defined an optimizer as the efficient stochastic gradient descent algorithm "adam" that means "Adaptive moment estimation". This is a popular version of gradient descent because it automatically tunes itself and gives good results in a wide range of problems.

It was first introduced in 2005 by Diederik Kingma and Jimmy Ba specifying that the Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing.

## 6.5. Train Keras Model

Once the model is compiled and has an efficient computation, it should be ready to be executed on some data.

A simple way to train the model is to call the fit () function on the model.

Training occurs epochs and each epoch is split into batches. In fact, one epoch is composed of one or more batches, based on the chosen batch size and the model is fit for many epochs.

Epoch: One pass through all of the rows in the training dataset. It is the number of epochs to train the model. An epoch is an iteration over the entire input and output data provided.

Batch: One or more samples considered by the model within an epoch before weights are updated. It is the number of samples per gradient update. If unspecified, batch_size will default to 32.

The training process will run for a fixed number of iterations through the dataset called epochs, that we must specify using the epochs argument. Another parameter must be set and it is the number of dataset rows that are considered before the model weights are updated within each epoch, called the batch size and set using the batch_size argument. These configurations can be chosen experimentally by trial and error.

The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters.

When all training samples are used to create one batch, the learning algorithm is called batch gradient descent. When the batch is the size of one sample, the learning algorithm is called stochastic gradient descent. When the batch size is more than one sample and less than the size of the training dataset, the learning algorithm is called mini-batch gradient descent.

- Batch Gradient Descent. Batch Size = Size of Training Set
- Stochastic Gradient Descent. Batch Size = 1
- Mini-Batch Gradient Descent. 1 < Batch Size < Size of Training Set

The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. In other words, it is the number of complete passes through the training dataset before the training process is terminated.

One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters.

## 6.6. Test Keras Model

Now that the neural network has been trained on the entire dataset, it is possible to evaluate its performances on the same dataset.

This phase is written through the predict () function on the model that passes the same input and output used to train the model. This will generate a prediction for each input and output pair and collect scores.

Making predictions is as easy as calling the predict () function on the model. After the model is fit, predictions are made for all examples in the dataset.

All neural network models have two hidden layers of 100 nodes. The sigmoid activation function is chosen for the output layer. In this work, two different activation function has been used. Initially the Sigmoid function as the default one and then the ‘Relu? To see the differences in terms of loss.

# 7. Results

In this chapter the results of this work will be presented in the form of graph for different cases. After running several models with different hyperparameters, the results are shown as learning curves and error tables.

A logic sequence has been followed in order to choose the kind of algorithm and its parameters in the most efficient time and computational cost.

All the results units are m for displacements and Pa for tractions.

## 7.1 Dense

The dense model works without memory so it is not able to predict well elasto-plastic behaviour of granular materials in the training and testing phases.

First of all, after setting all parameters, it is possible to see differences in the graphs while changing the number epochs of the model. By increasing the epochs, the results will be more accurate and the error will decrease.

The system to define whether changing the parameters helps the prediction of the tractions is by looking at the overall error and how close the prediction curve becomes regarding the data curve.

In this way the distance between the effective and physical data and the computational predictions, known as loss is decreasing.

## 7.1.1. Changing Epochs

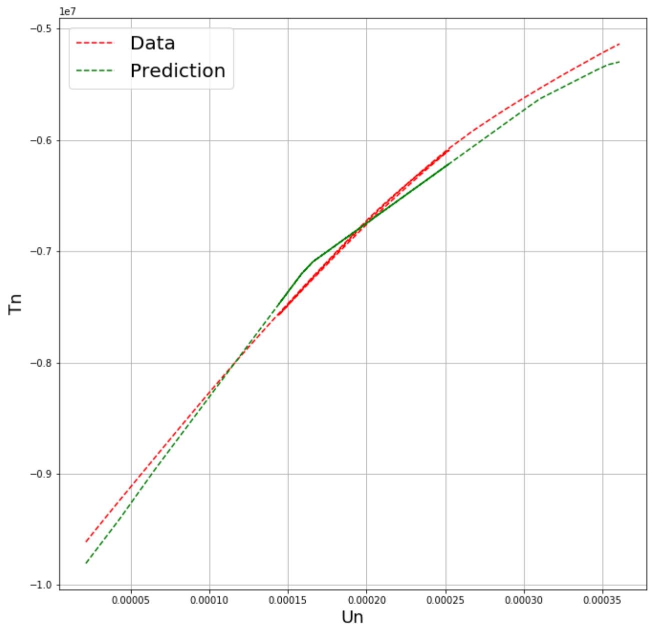


*Figure 7.1 Case 20, epochs 100, batch 100*

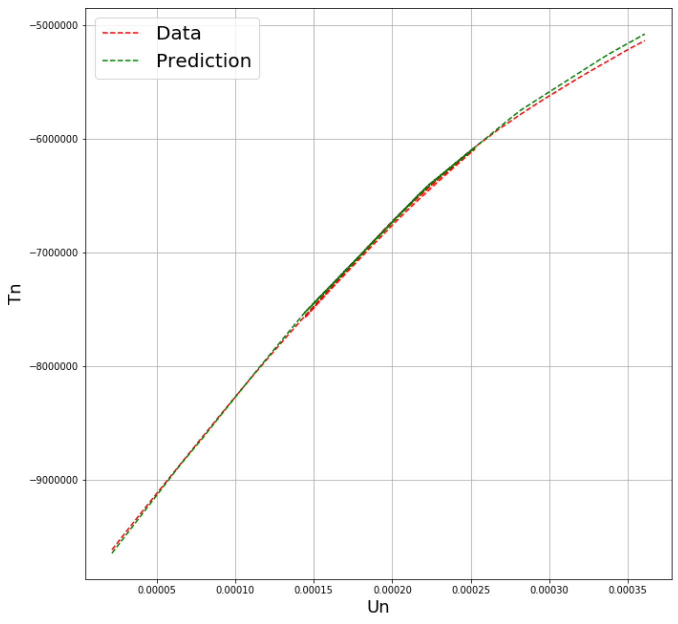


*Figure 7.2 Case 20, epochs 1000, batch 100*

In Fig. 7.1 and 7.2 the same case (case 20) is shown with different number of epochs. If the number of iteration increases, the curve that fit and predict the data will improve as noticeable in the two graphs.

### 7.1.2. Changing input

In this subchapter will be studied how the predictions improve while changing the number of input.

In the Fig. 7.3 and 7.4 that represent Case 1, the epochs are 1000 and batch size is 10. A notable improvement of the results is seen as porosity and the fabric tensor are added as input. Note that Fig. 7.3 has just two input (Un and Us) while Fig. 7.4 has 4 input (Un, Us, Porosity and Fabric Tensor).

Not every material property has the same value as input. In this case, by adding all the one available in the DEM simulation except from the coordination number, the dense model is capable to fit even the plasticity parts.

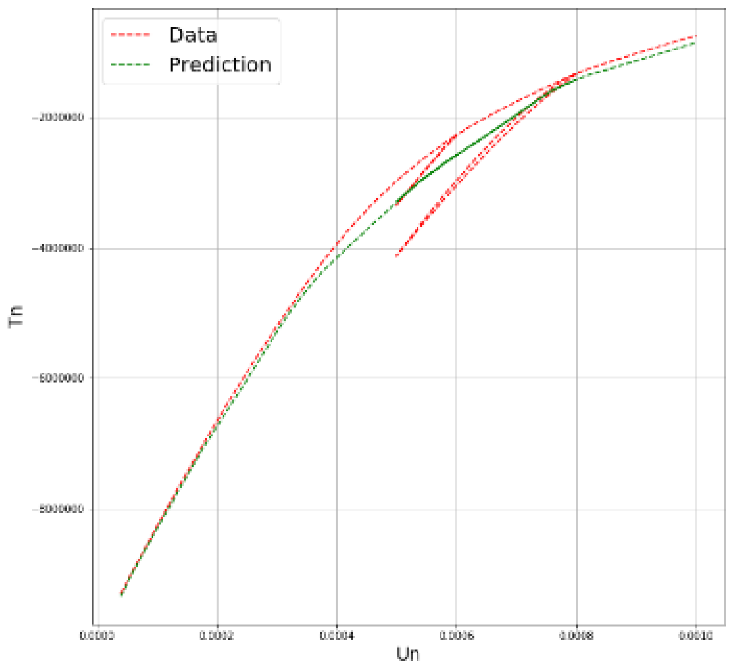


*Figure 7.3 Case 1, epochs 1000, batch 10 , input just displacement*

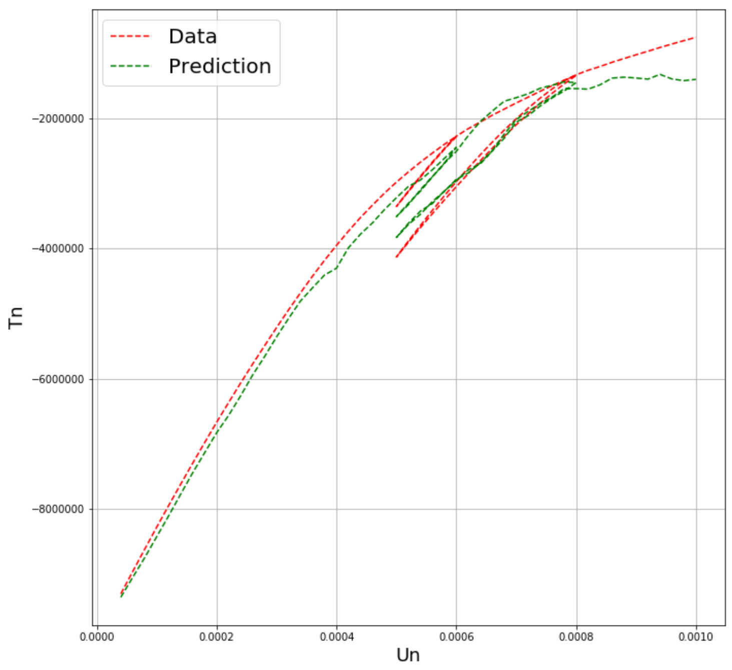


*Figure 7.4 Case 1, epochs 1000, batch 100, Porosity and Fabric Tensor*

It is also possible to see in these two figures that the Dense model is still roughly capable to predict more complicated curves thanks to the parameter Fabric Tensor.

This improvement is more evident in case 10 where the Fabric Tensor changes completely the prediction curve and the result is more accurate but still non-sufficient to predict well the problem:

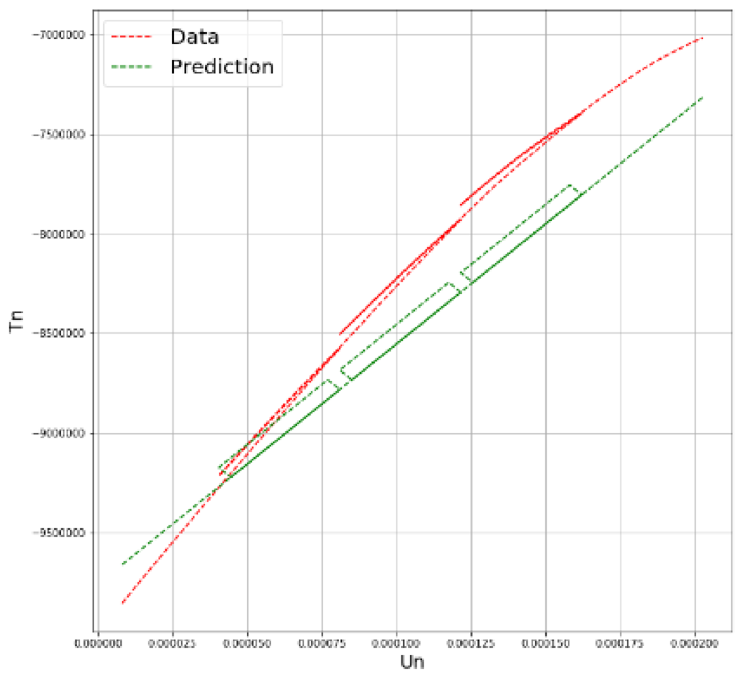


*Figure 7.5 Case 10, epochs 1000, batch 10, Input: Un, Us and  Porosity*

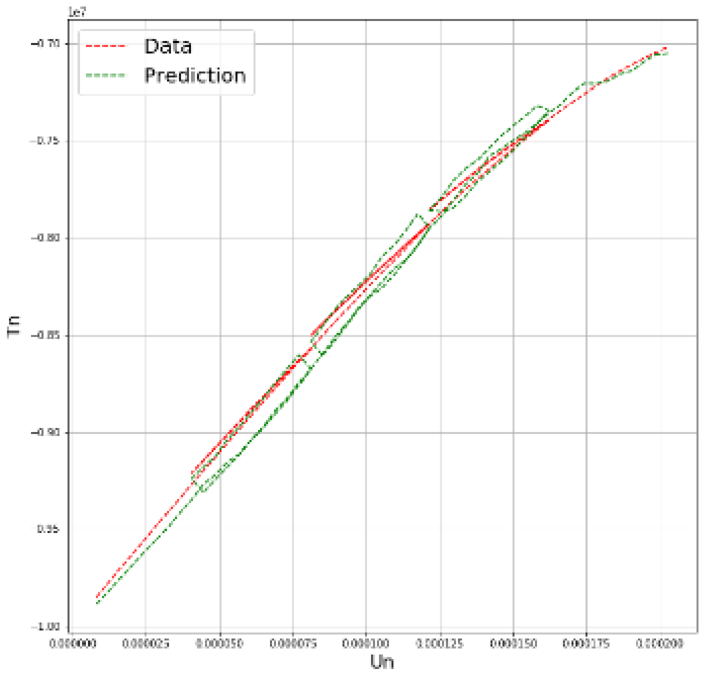


*Figure 7.6 Case 10, epochs 1000, batch 10, Input: Un, Us, Porosity and Fabric Tensor*

In conclusion, the dense model even increasing the number of input or the iterations, is not capable to predict the non-elastic curves as seen in the previous images.

The main reason could be that the Dense layer has not been developed to perform with a large and sophisticated amount of data.

## 7.2 LSTM

The LSTM layers as well as the GRU in the next subchapter, are capable to better capture the elasto-plastic behaviour of granular materials thanks to the memory gates.It is also commonly used for manging thousands of data with different relationships and pattens among them.

In this subchapter the best combination of input is studied with 100 epochs because the computational cost is lesser that increasing the iterations. Then, the number of iterations will be increases with the case of input with less error in order to perform a more efficient analysis. The error will be computed with the mean squared error between the training and the testing data but also in the form of learning curves.

The data obtained from lower-scale numerical simulations are pre-processed and converted to specific data structure compatible with the LSTM and GRU training and validation algorithms.

### 7.2.1. Changing input

The best combination of input has been studied in terms of error. The general loss is calculated from the training dataset that gives an idea of how well the model is learning. The other type of error is the validation loss that is calculated from a hold-out validation dataset that gives an idea of how well the model is generalizing.

The 'validation split' command is a float between 0 and 1. Basically, it is a fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch. The validation data is selected from the last samples in the input and output data provided, before shuffling.

Among all of the input data, the 90% of them are used to train the neural network and the 10% is used to validate it. The loss is the mean squared error between the dataset and the data trained/validated.

Here our goal is to check whether the incorporation of any of these additional data as input in the RNN network improves the prediction quality.

Running the neural network with 100 epochs, the results in terms of error at the 100th epoch are:

*Table 7.1 Errors Changing the number of inputs (LSTM)*

| *Input* | *Training loss* | *Validation loss* |
|---|---|---|
| *Us, Un* | $4.34*10^{-4}$ | $2.94*10^{-4}$ |
| *Us, Un, Coordination number* | $3.90*10^{-4}$ | $2.65*10^{-4}$ |
| *Us, Un, Porosity* | $4.19*10^{-4}$ | $2.86*10^{-4}$ |

| | | |
|---|---|---|
| *Us, Un, Fabric Tensor* | $6.14*10^{-5}$ | $5.60*10^{-5}$ |
| *Us, Un, Coordination number and porosity* | $3.91*10^{-4}$ | $2.65*10^{-4}$ |
| *Us, Un, Coordination number and Fabric Tensor* | $6.23*10^{-5}$ | $5.93*10^{-5}$ |
| *Us, Un, Porosity and Fabric Tensor* | $6.07*10^{-5}$ | $5.52*10^{-5}$ |
| *Us, Un, Coordination number, porosity and Fabric Tensor* | $5.81*10^{-5}$ | $5.22*10^{-5}$ |

The fabric tensor is the parameter that better helps the neural network to predict the tractions. The error is one order smaller that with just the other parameters.

It is possible to see that the coordination number is more effective than the porosity. Computationally speaking, this means that the fabric tensor as well as the coordination number, better represents the behaviour of a granular material.

In terms of error, the best combination is the one with all of the parameters as input.

In the following images it is possible to see the differences in term of predicting curves with the number of inputs in case 200 with 100 epochs.

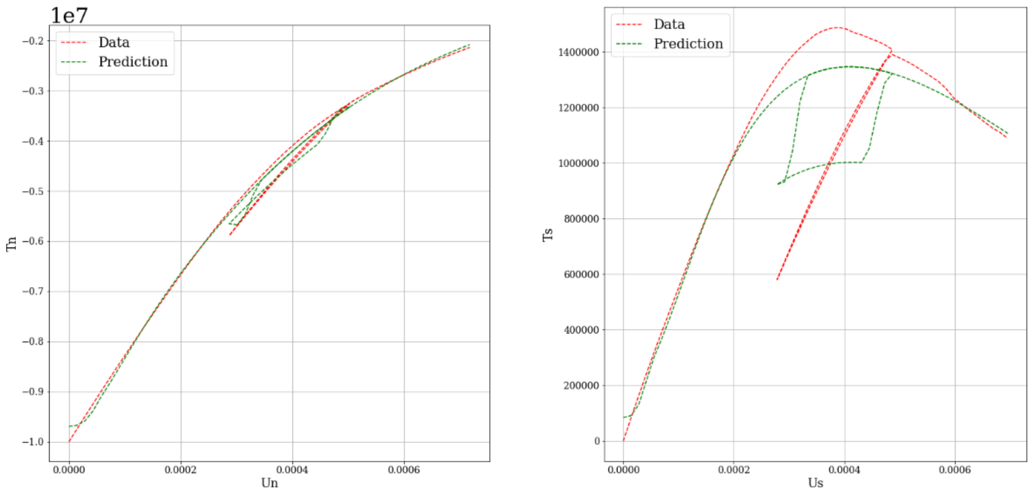


*Figure 7.7 Case 200 Input: Un, Us*

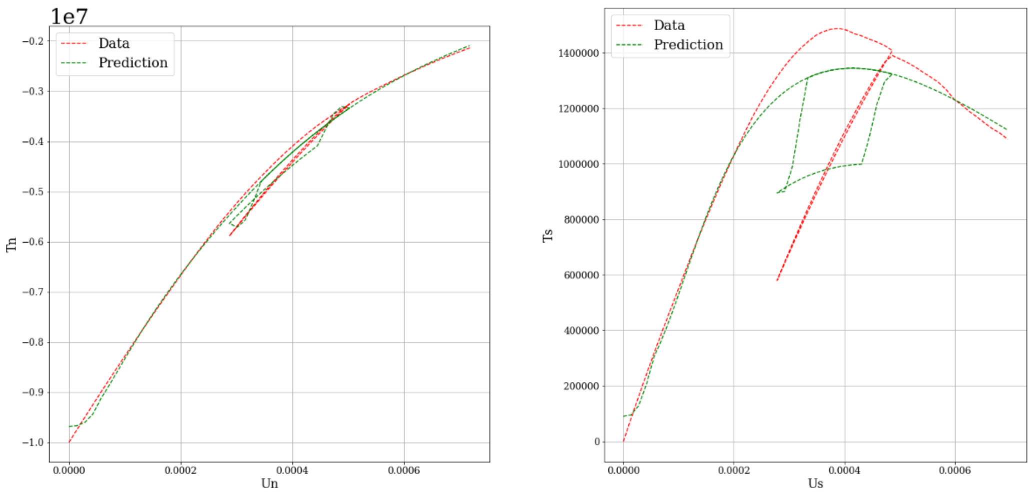


*Figure 7.8 Case 200 Input: Un, Us and Porosity*

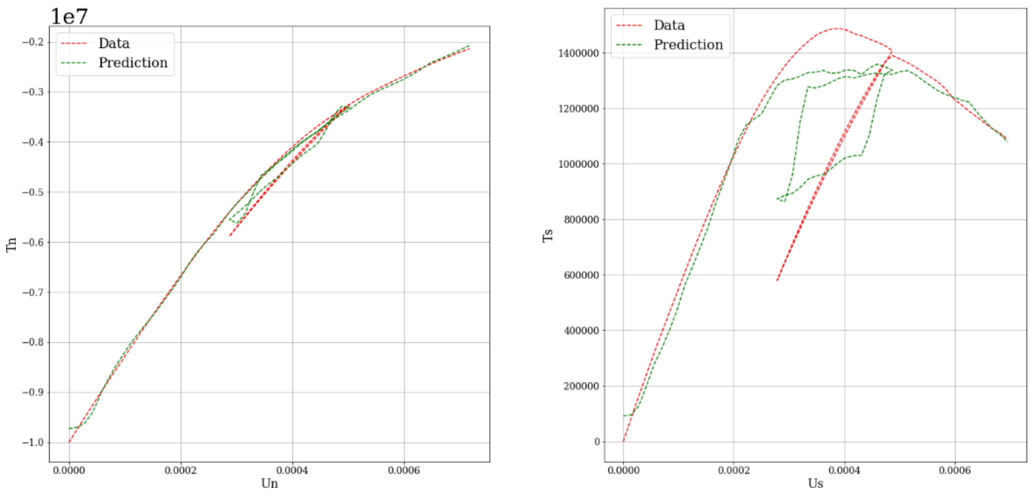


*Figure 7.9 Case 200 Input: Un, Us and Coordination number*

Fig. 7.8 and 7.9 have the same number of input but one of them seems to have more influence in the prediction curve, even if in this case is not well fitted.

The coordination number generally decrease the error and helps the curve to become closer to Case 200.

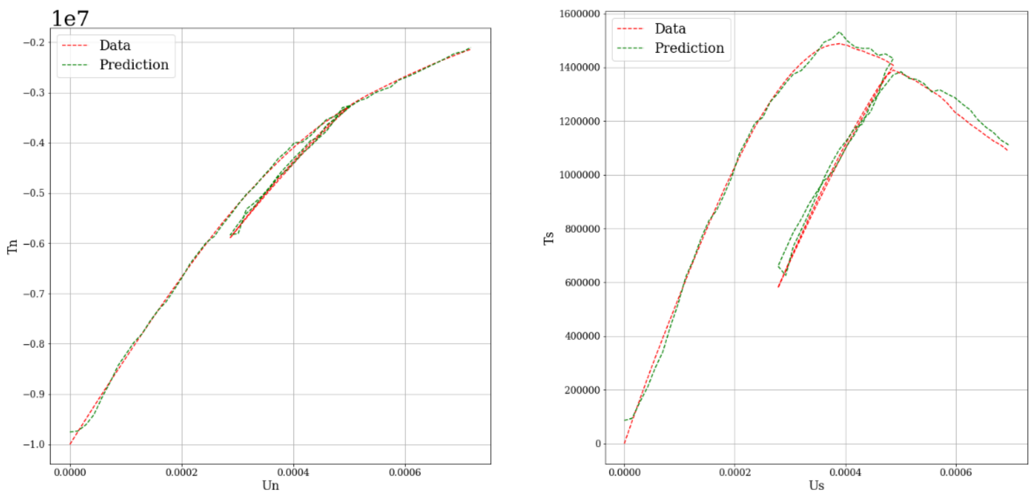


*Figure 7.10 Case 200 Input: Un, Us and Fabric Tensor*

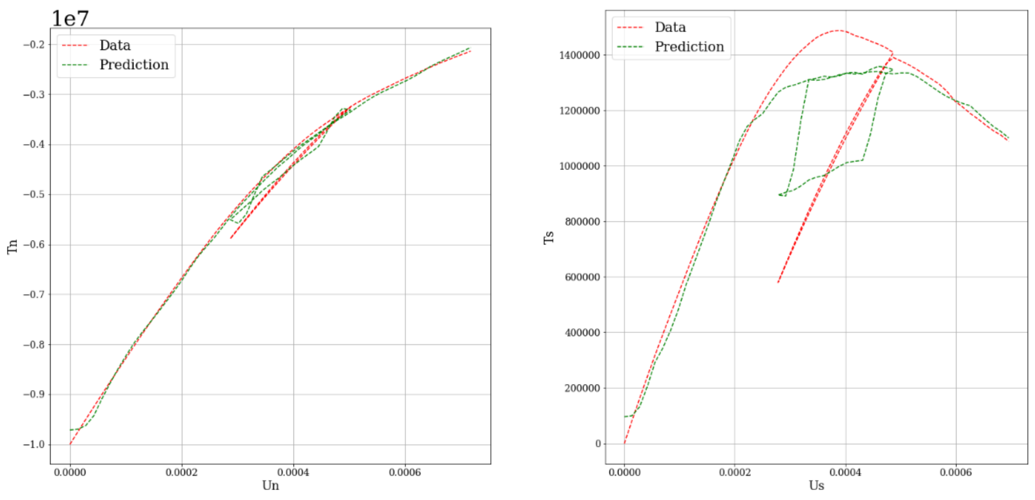


*Figure 7.11 Case 200 Input: Un, Us Porosity and Coordination number*

Even with porosity and coordination number together, an impressive improvement is due to the fabric tensor in Fig. 7.10

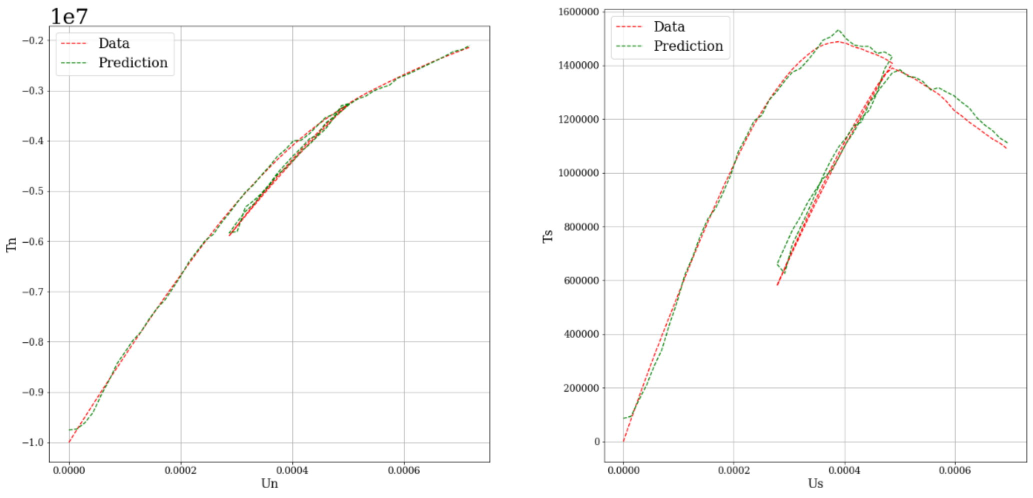


*Figure 7.12 Case 200 Input: Un, Us, Coordination number and Fabric Tensor*

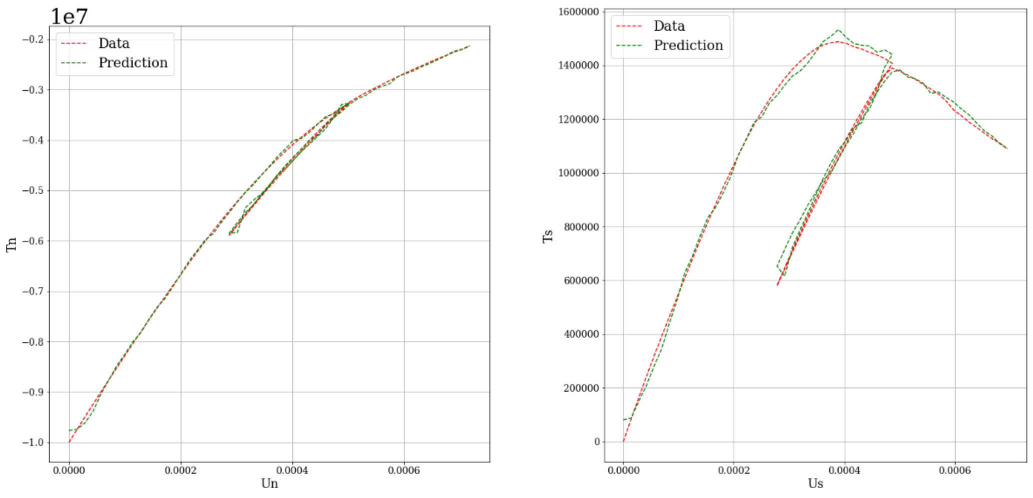


*Figure 7.13 Case 200 Input: Un, Us, Porosity and Fabric Tensor*

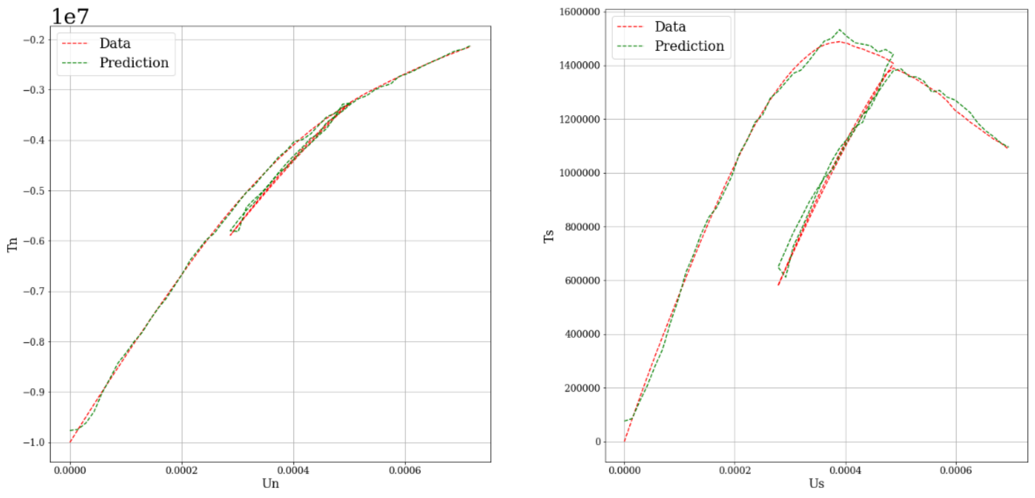


*Figure 7.14 Case 200 Input: Un, Us, Coordination number, Porosity and Fabric Tensor*

In conclusion, the best combination of input is the one reported in Fig. 7.14 with all the input together. The error is lesser than all the other combinations.

## 7.2.2. Learning curves

Generally, a learning curve is a plot that shows time or experience on the x-axis and learning or improvement on the y-axis. During the training of a machine learning model, the current state of the model at each step of the training algorithm can be evaluated. It can be evaluated on the training dataset to give an idea of how well the model is "*learning*." It can also be evaluated on a hold-out validation dataset that is not part of the training dataset. Evaluation on the validation dataset gives an idea of how well the model is "*generalizing*."

- Train Learning Curve: Learning curve calculated from the training dataset that gives an idea of how well the model is learning.
- Validation Learning Curve: Learning curve calculated from a hold-out validation dataset that gives an idea of how well the model is generalizing.

It is common to create dual learning curves for a machine learning model during training on both the training and validation datasets.

The curves in this work are optimization learning curves where the curves are calculated on the metric by which the parameters of the model are being optimized e.g. loss.

A good fit is the goal of the learning algorithm and exists between an overfit and underfit model.

A good fit is identified by a training and validation loss that decreases to a point of stability with a minimal gap between the two final loss values.

The loss of the model will almost always be lower on the training dataset than the validation dataset. This means that we should expect some gap between the train and validation loss learning curves. This gap is referred to as the “generalization gap.”

A plot of learning curves shows a good fit if:

- The plot of training loss decreases to a point of stability.
- The plot of validation loss decreases to a point of stability and has a small gap with the training loss.

Here are presented three different learning curves, depending on the number of epochs they are referred to. All three Figures are referred to the model with all the inputs on it because it is the best fit with less error.

In general, the models show a good fit and there is not an overfitting problem. The curves from Fig, 7.15 to 7.17 show a slightly progress with the increasing of the number of epochs.

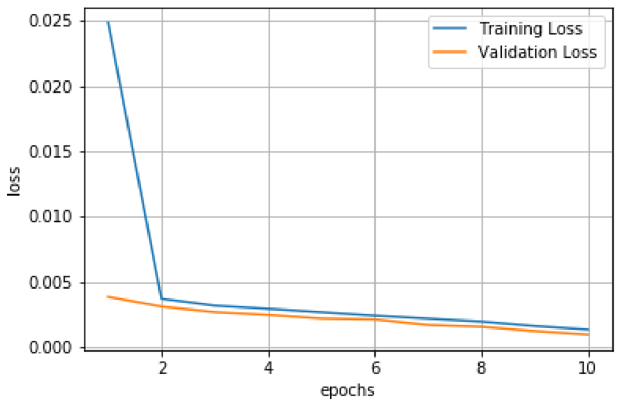


*Figure 7.15 Learning curves with 10 epochs*

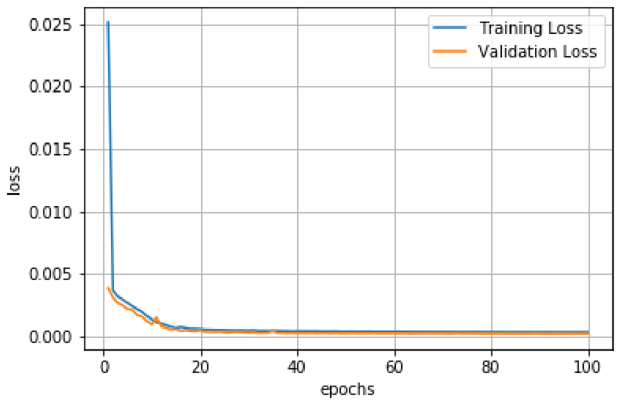


*Figure 7.16 Learning curves with 100 epochs*

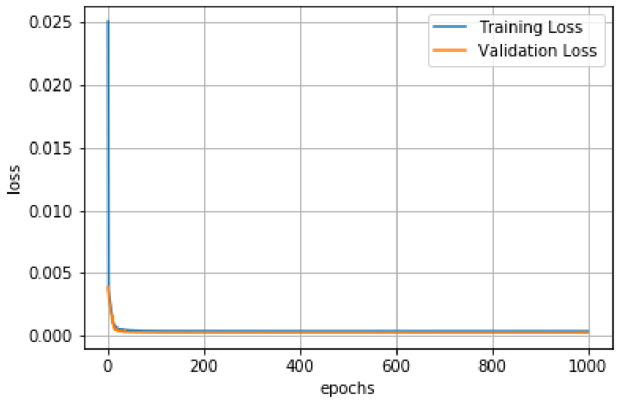


*Figure 7.17 Learning curves with 1000 epochs*

## 7.2.3. Changing Epochs

The error between the train and the test phase is valuated with the mean squared error. In this work the error is divided in each case of the dataset, the first 10000 rows of the csv file are used to train the network while the other to test the model.

The error decrease with the number of epochs as seen in §7.2.2 and the model used in this subchapter has all of the input in order to minimize the error.

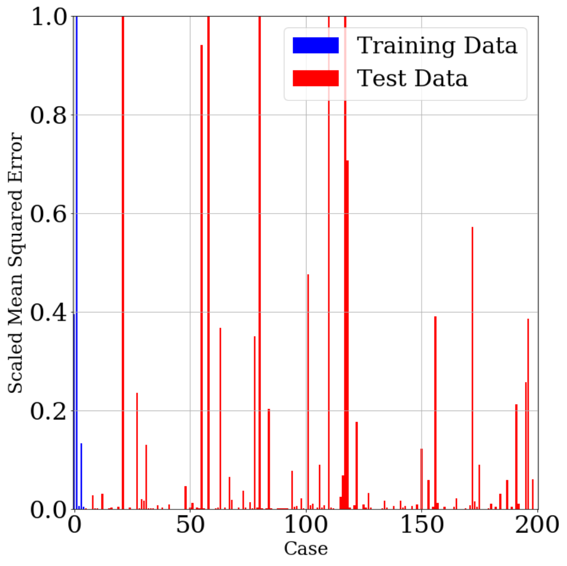


*Figure 7.18 Mean Squared Error with 10 epochs*

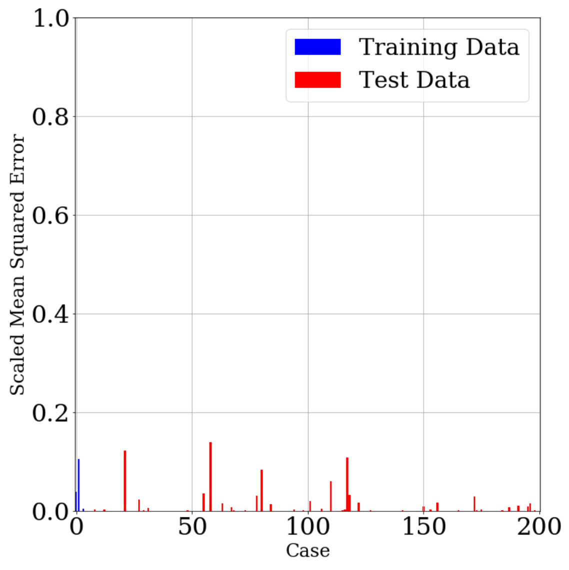


*Figure 7.19 Mean Squared Error with 100 epochs*

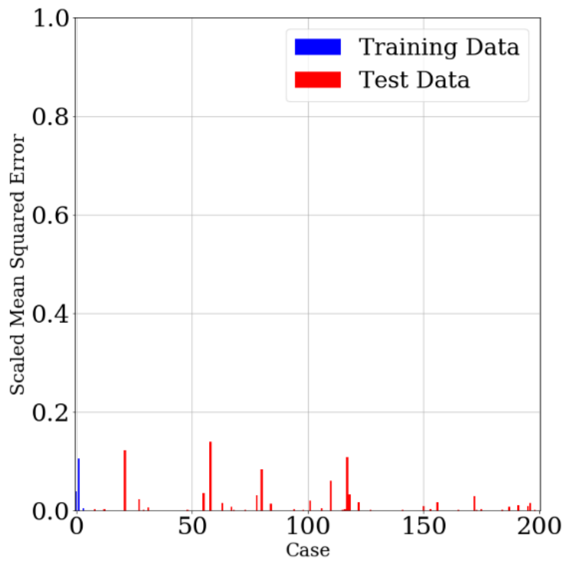


*Figure 7.20 Mean Squared Error with 1000 epochs*

Every case has a distinct error because some cases are different and the related curves have various shapes among the other. This means that the model cannot predict well some of them. There are a lot of improvement between Fig. 7.18 and 7.19 because the model can predict much more data. From fig. 7.19 and 7.20 there are no big differences.

With the best combination of input found in the previous subchapter, it is possible to increase the epochs and see the loss for training and validation at the 1000th epoch.

*Table 7.2 Errors changing the number of epochs (LSTM)*

| *Total epochs* | *Training loss* | *Validation loss* |
|---|---|---|
| *100* | $5.819*10^{-5}$ | $5.22*10^{-5}$ |
| *1000* | $5.14*10^{-5}$ | $4.62*10^{-5}$ |

Now will be presented the differences between some cases with respectively 100 and 1000 epochs. Note that the first case has no component in s direction.

- Case 1

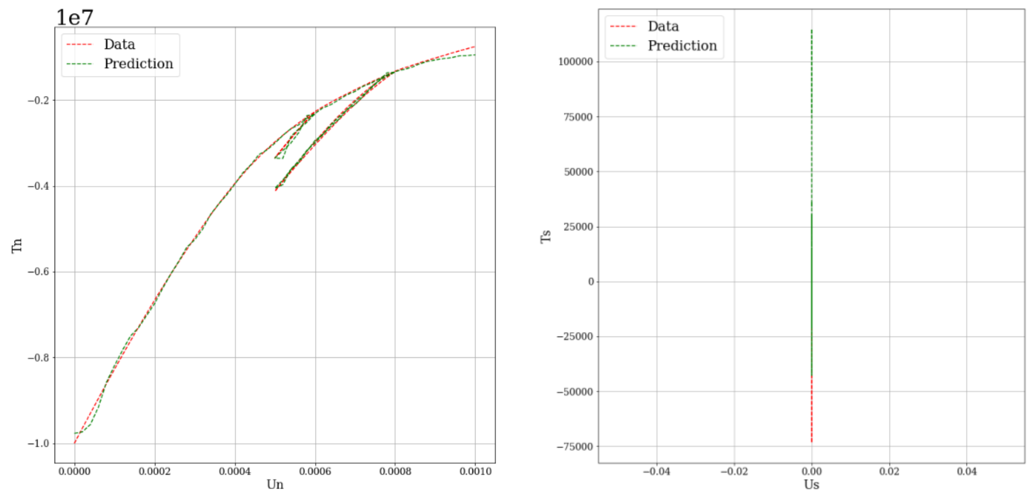


*Figure 7.21 Case 1, 100 epochs*

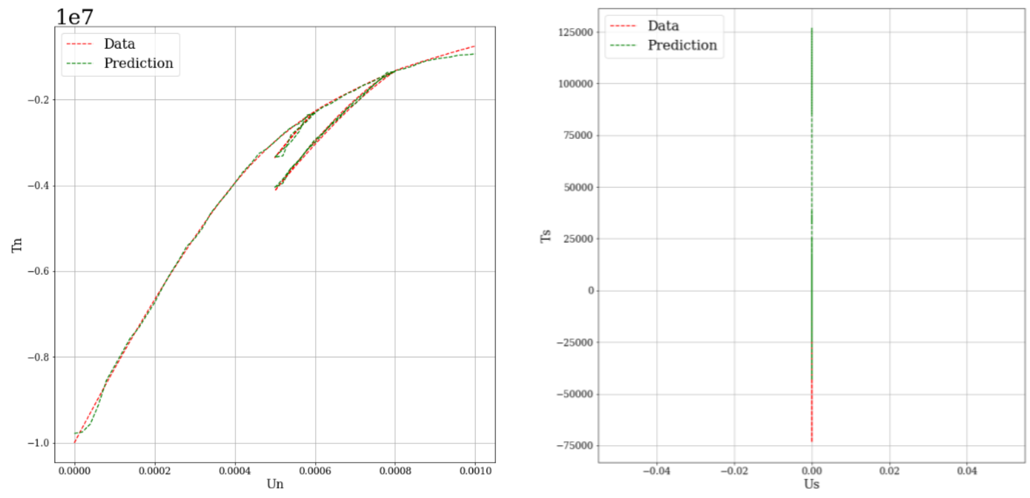


*Figure 7.22 Case 1, 1000 epochs*

- Case 2

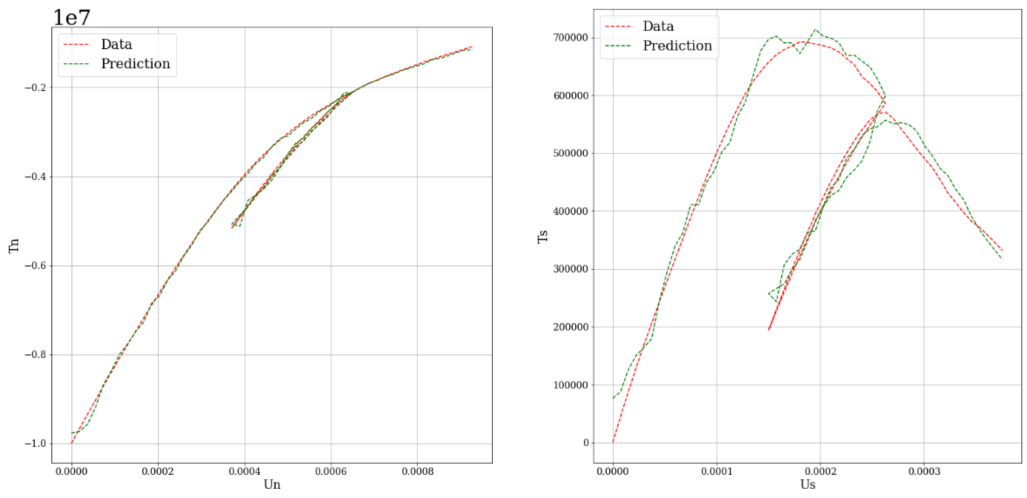


*Figure 7.23 Case 2, 100 epochs*

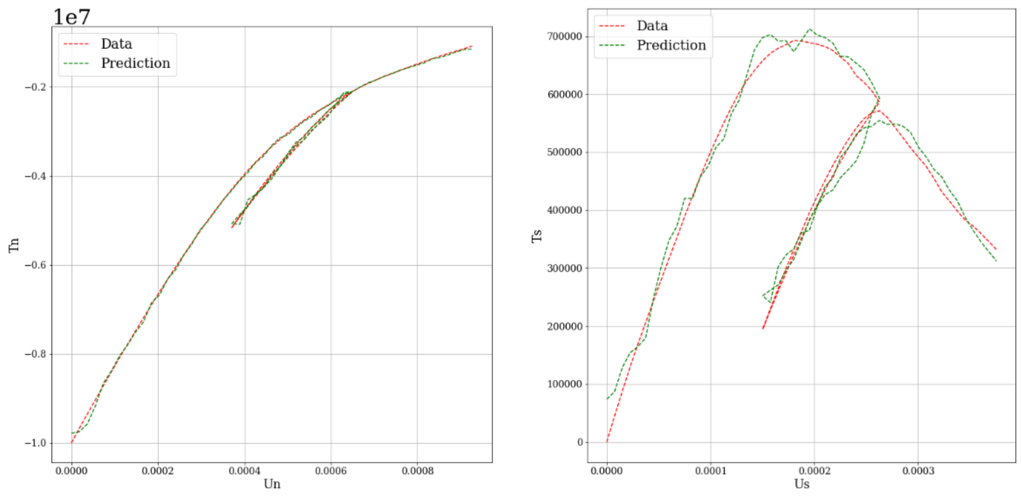


*Figure 7.24 Case 2, 1000 epochs*

- Case 10

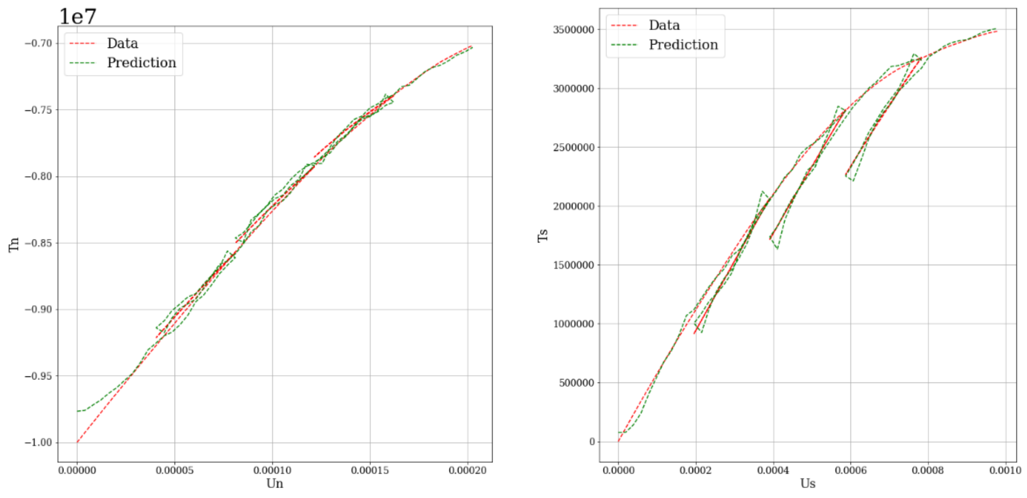


*Figure 7.25 Case 10, 100 epochs*

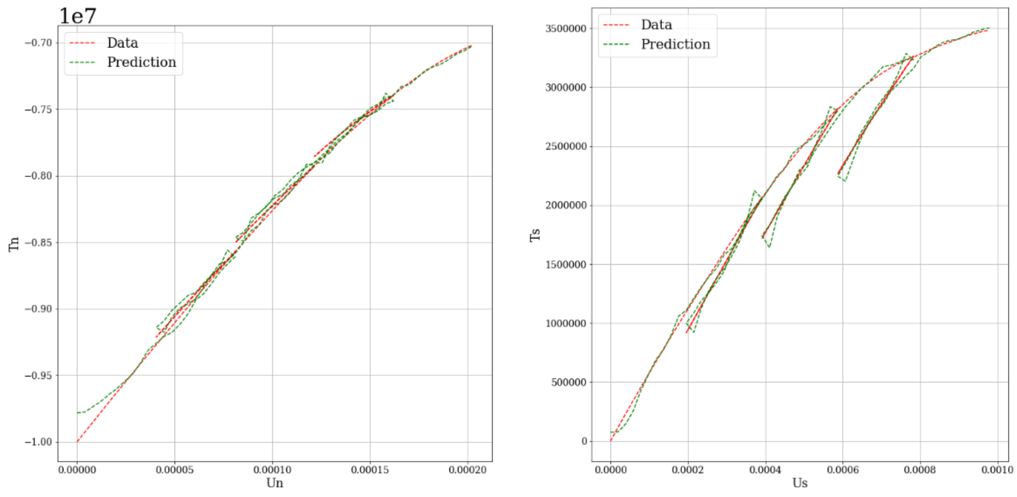


*Figure 7.26 Case 10, 1000 epochs*

- Case 20

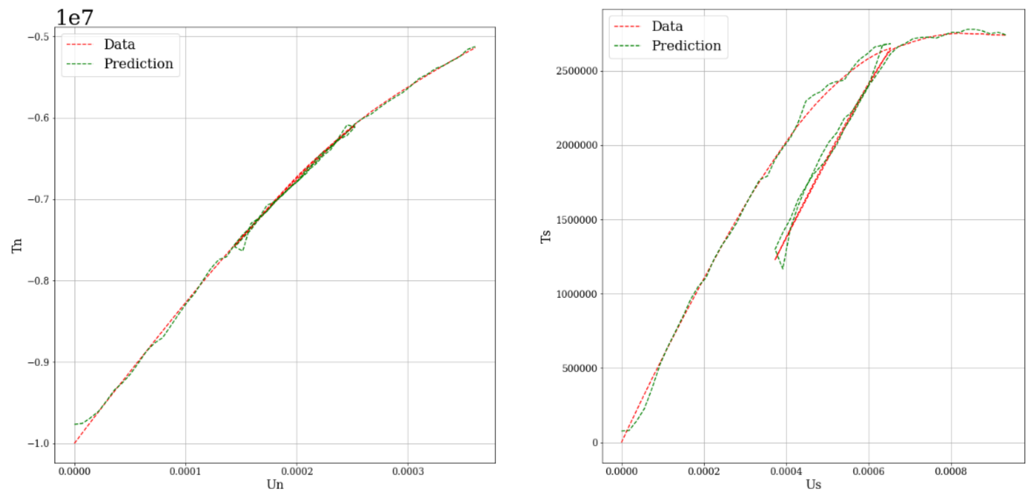


*Figure 7.27 Case 20, 100 epochs*

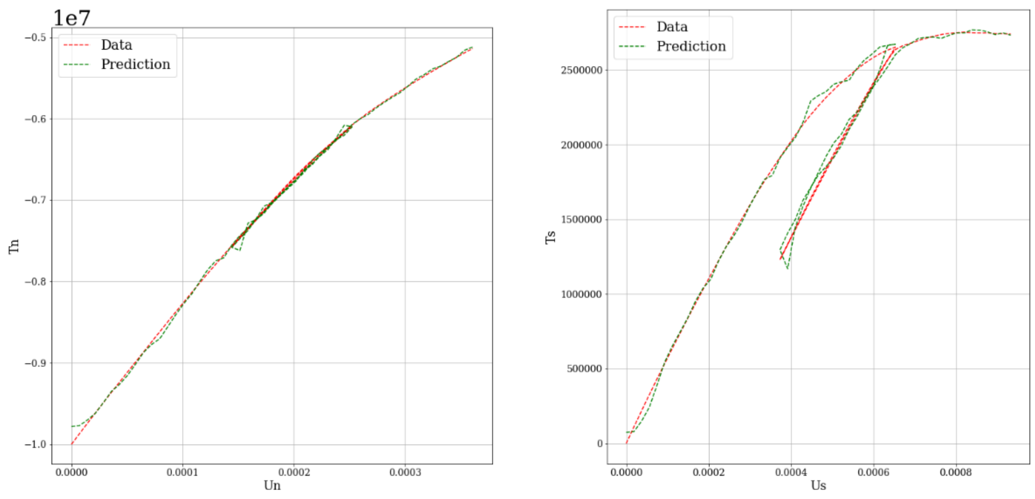


*Figure 7.28 Case 20, 1000 epochs*

- Case 50

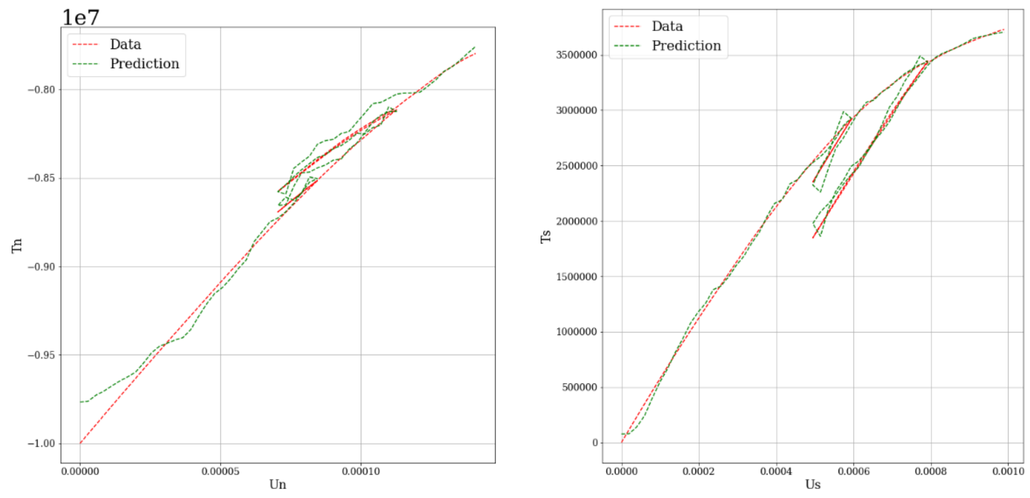


*Figure 7.29 Case 50, 100 epochs*

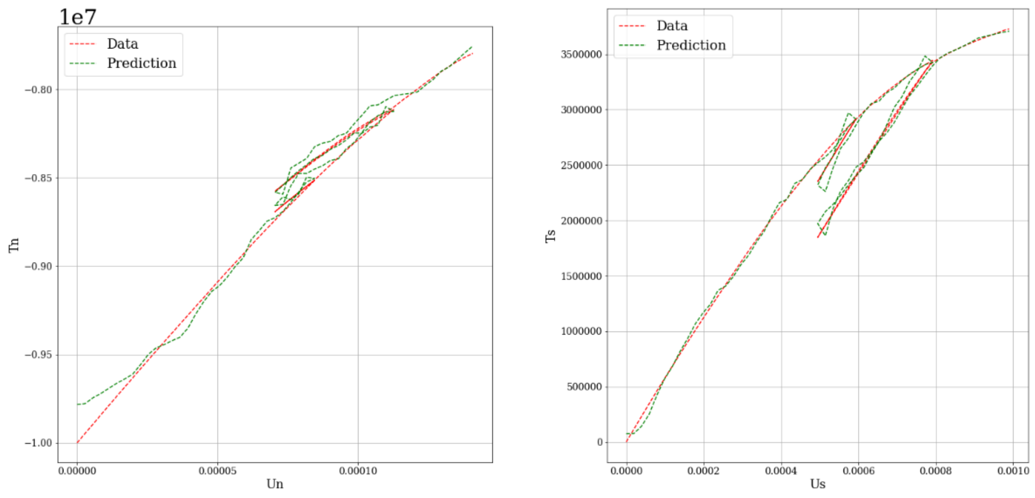


*Figure 7.30 Case 50, 1000 epochs*

- Case 100

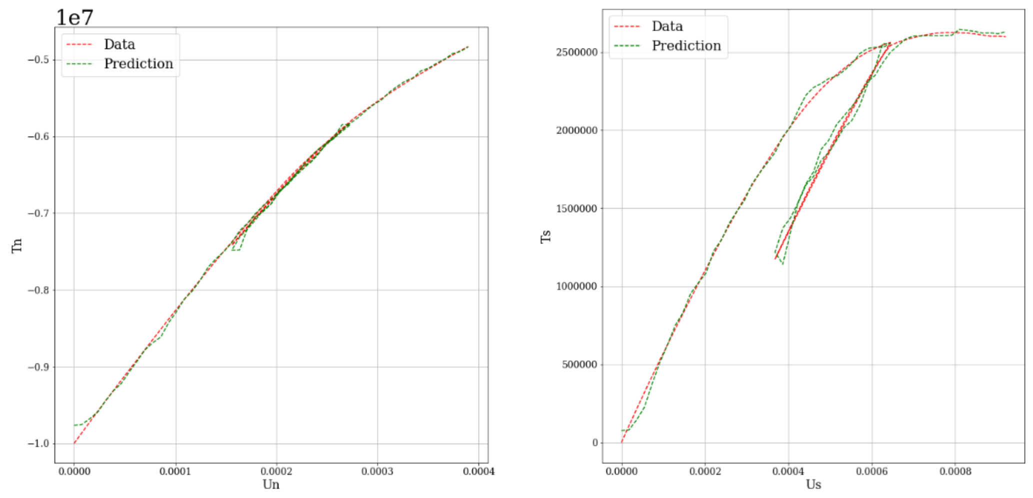


*Figure 7.31 Case 100, 100 epochs*

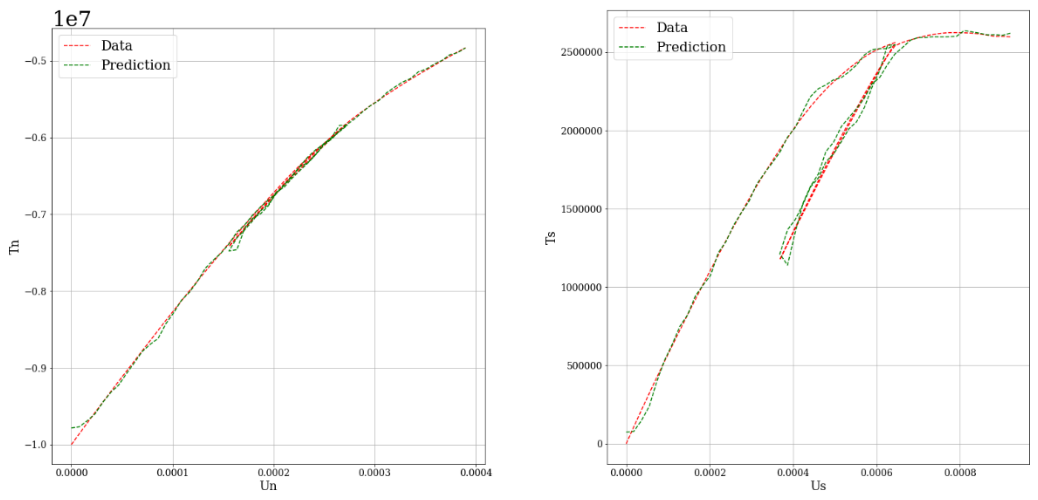


*Figure 7.32 Case 100, 1000 epochs*

- Case 200

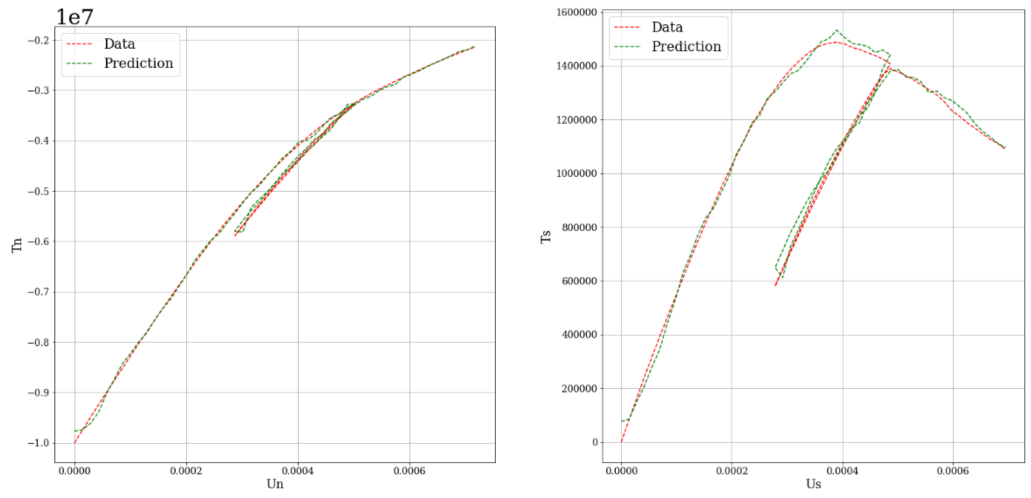


*Figure 7.33 Case 200, 100 epochs*

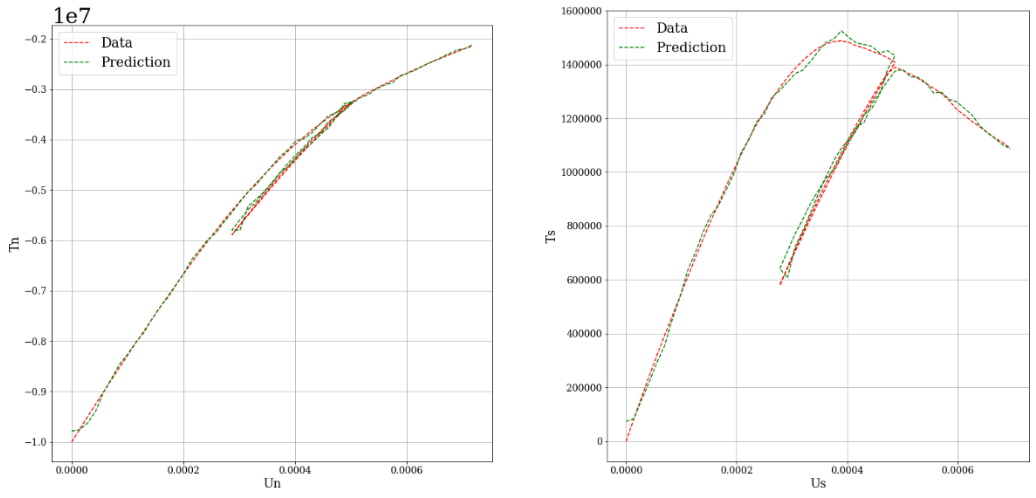


*Figure 7.34 Case 200, 1000 epochs*

### 7.2.4. Changing activation function

The differences between two main activation functions have been developed in this work. All of the previous models had the "*Sigmoid function*" as activation function and the other one to code was the "*Relu"* function.

Basically, the only difference in practice is in term of the validation loss because it doesn't change the general loss at the 1000th epoch.

*Table 7.3 Errors changing activation function (LSTM)*

| *Activation function* | *Training loss* | *Validation loss* |
|---|---|---|
| *Sigmoid* | $5.14*10^{-5}$ | $4.62*10^{-5}$ |
| *Relu* | $5.14*10^{-5}$ | $4.49*10^{-5}$ |

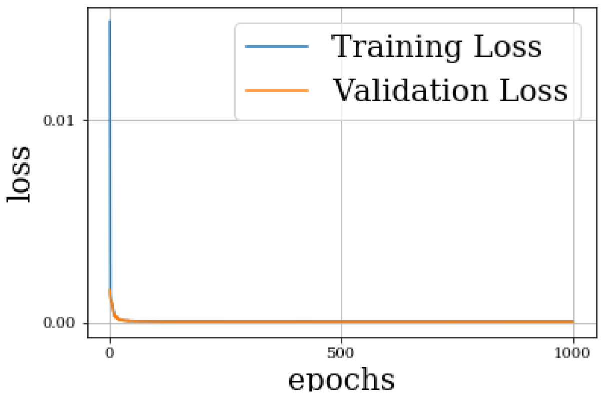


*Figure 7.35 learning curve with Relu activation function*

As seen in Fig 7.35 in this model there are no problems of overfitting and in general the error can be approximated to zero.

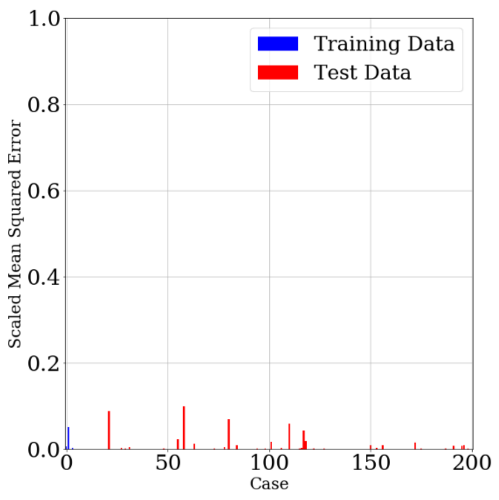


*Figure 7.36 Mean squared error-Relu activation function*

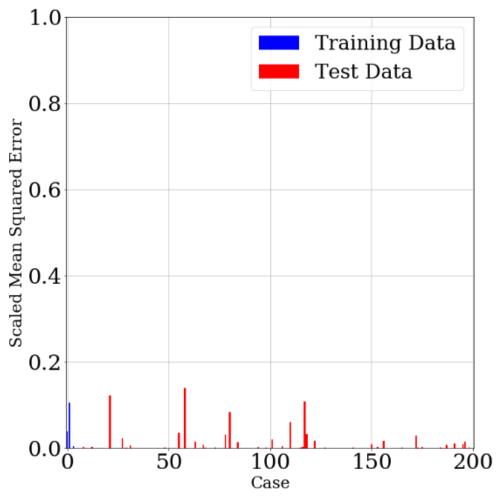


*Figure 7.37 Mean squared error-Sigmoid activation function*

In Fig. 7.36 the mean squared error associated to the model with the Relu function is lower to the one with the sigmoid function as seen in Fig. 7.37.

## 7.3 GRU

The GRU algorithm is a form of LSTM that has been modified as explained in the previous chapters. While all the parameters stays the same, the model with GRU layers doesn't predict well as the LSTM model so it has been necessary to reduce the batch size in order to capture more details in the traction-separations curves.

First of all it has been necessary to build a model made by two GRU layers with 2 neurons each and without any dropout. Another Dense layer has been added at the end.

All of the model had as input the displacements, the porosity, the coordination number and the fabric tensor. The Traction had been used as outputs.

### 7.3.1. Changing epochs

Differences in terms of learning rate between a model trained with 130 epochs such as the LSTM model and with the sigmoid activation function. The validation split is 0.1 as the previous cases.

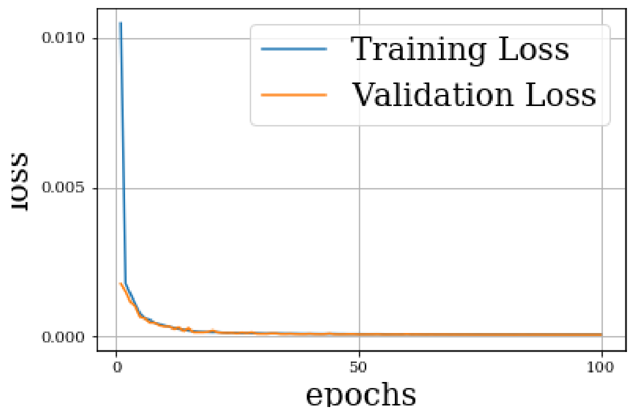


*Figure 7.38 Learning curves with 100 epochs*

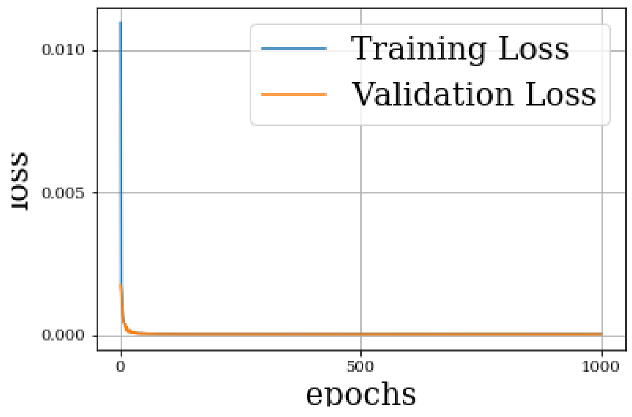


*Figure 7.39 Learning curves with 1000 epochs*

The error has decreased but not considerably between the two cases at the last epoch. Note that there are no problems of overfitting so the model is training well.

*Table 7.4 Errors changing the number of epochs (GRU)*

| Epoch | Training loss | Validation loss |
|---|---|---|
| 100 | $5.75*10^{-5}$ | $5.04*10^{-5}$ |
| 1000 | $5.17*10^{-5}$ | $4.57*10^{-5}$ |

Some cases will now be presented as comparison between these two situations.

- Case 1

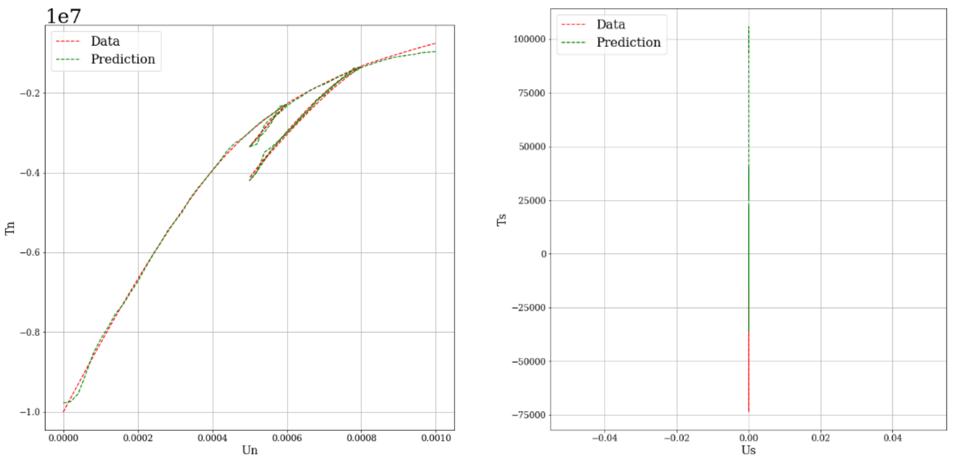


*Figure 7.40 Case 1, 100 epochs*

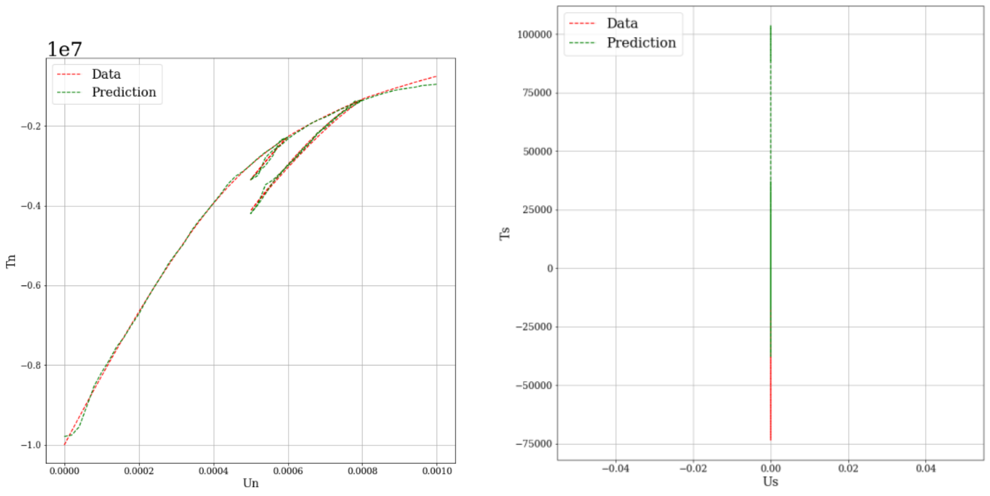


*Figure 7.41 Case 1, 1000 epochs*

- Case 50

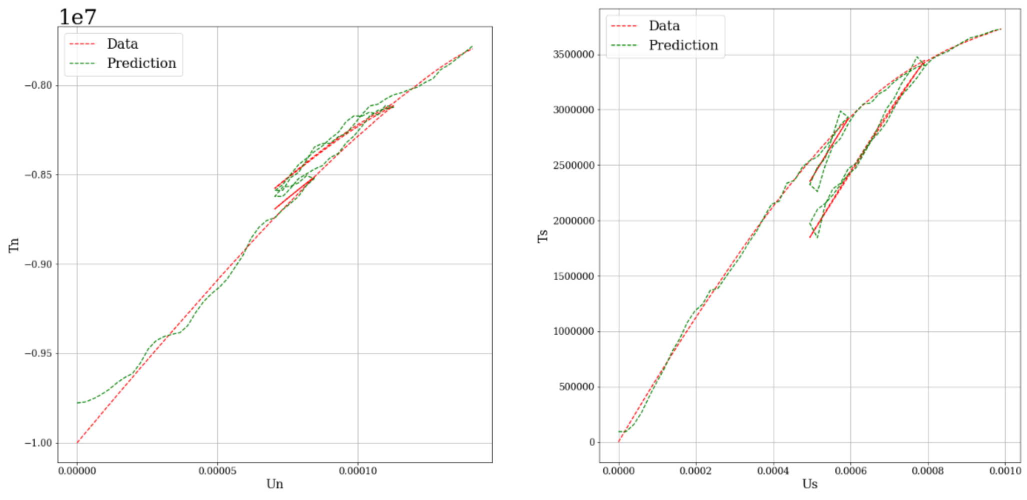


*Figure 7.42 Case 50, 100 epochs*

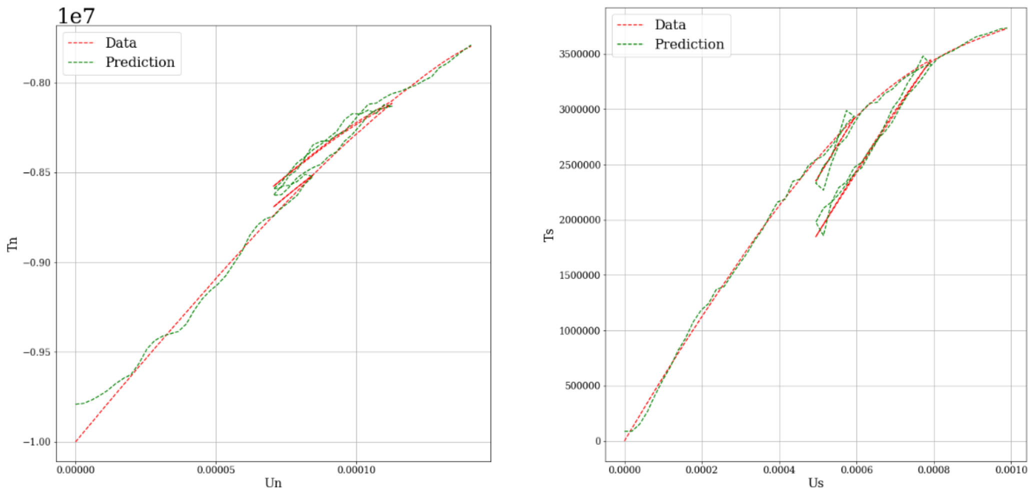


*Figure 7.43 Case 50, 1000 epochs*

- Case 200

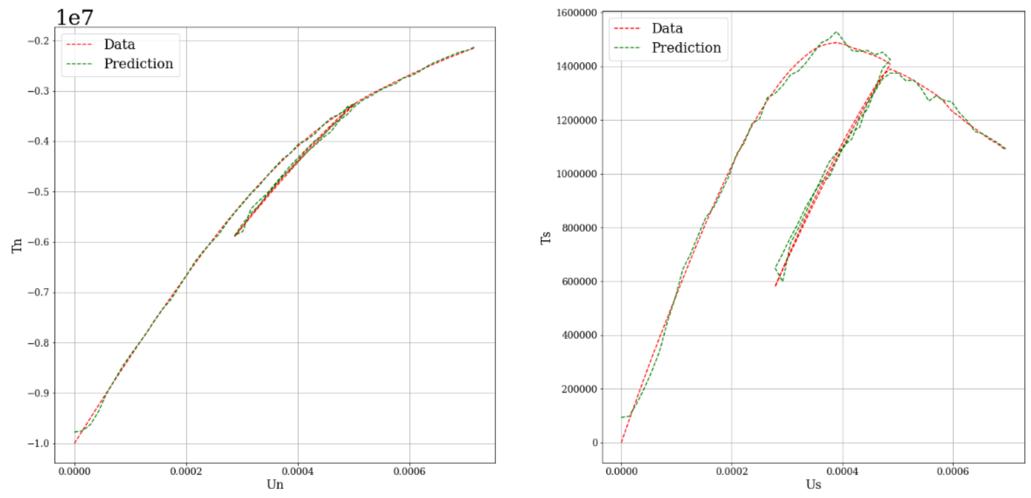


*Figure 7.44 Case 200, 100 epochs*

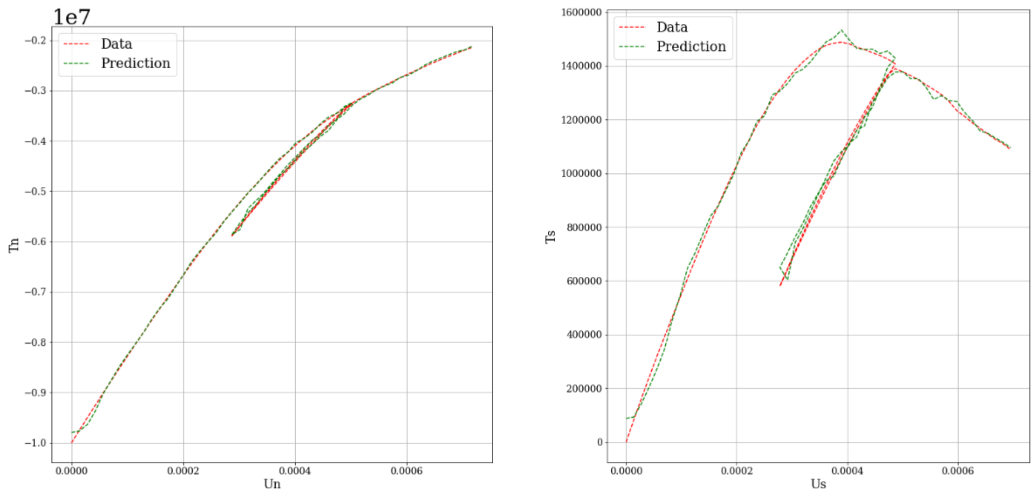


*Figure 7.45 Case 200, 1000 epochs*

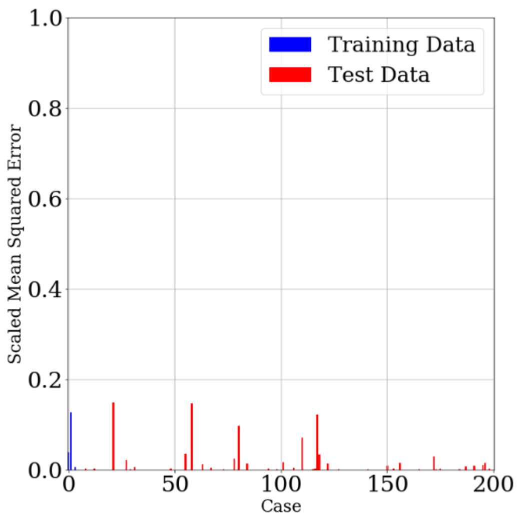


*Figure 7.46 Mean Squared Error, 100 epochs*

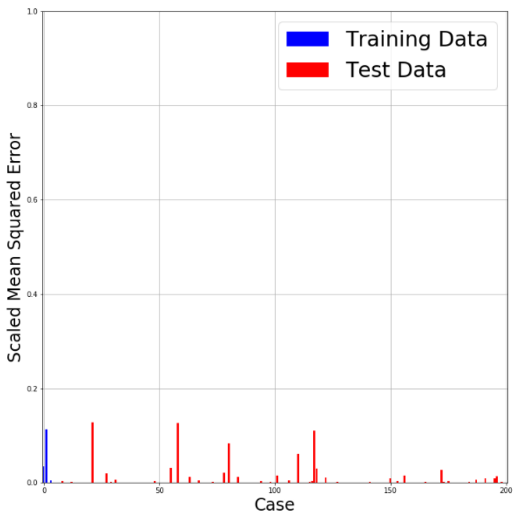


*Figure 7.47 Mean Squared Error, 1000 epochs*

Since increasing the number of epochs has not improved the differences in terms of prediction curves and final error, the GRU model needed to be modified and adjusted to these kind of dataset so the first thing that has been changed is the activation function.

### 7.3.2. Changing activation function

In order to minimize the time and computational cost, the Relu activation function has been introduced in the model with 100 epochs.

In terms of error, the loss at the 100th epoch is:

Table 7.5 Errors changing the activation function (GRU)

| *Activation function* | *Training loss* | *Validation loss* |
|---|---|---|
| *Sigmoid* | $5.75*10^{-5}$ | $5.04*10^{-5}$ |
| *Relu* | $5.49*10^{-5}$ | $4.87*10^{-5}$ |

## 7.3.3. Changing batch size

Since the error in general can be improved, the batch size has been changed from 130 to 64. The proceed is based on trial and error although there are some algorithms that can help to decide with size is better. In general, choosing the batch size depend on the problem type, the size of the dataset and the layers. In this case, with GRU network a smaller batch size has been introduced to better capture the nonlinear behaviour of the material.

All the trained models in this subchapter have the Relu Activation function because it can reduce the error and have been trained with 100 epochs.

In Tab. 7.6 it is possible to see how the mean squared error in each phase of the neural network is reducing while changing the batch size but not the activation function.

Table 7.6 Errors changing batch size (GRU)

| *Batch size* | *Training loss* | *Validation loss* |
|---|---|---|
| *130* | $5.49*10^{-5}$ | $4.87*10^{-5}$ |
| *64* | $3.11*10^{-5}$ | $2.27*10^{-5}$ |

As last improvement the number of epochs has been increased to 1000 to see the real differences in terms of error and graphically in the traction-separation curves.

The Relu activation function and batch size of 64 samples has been maintained in the final model.

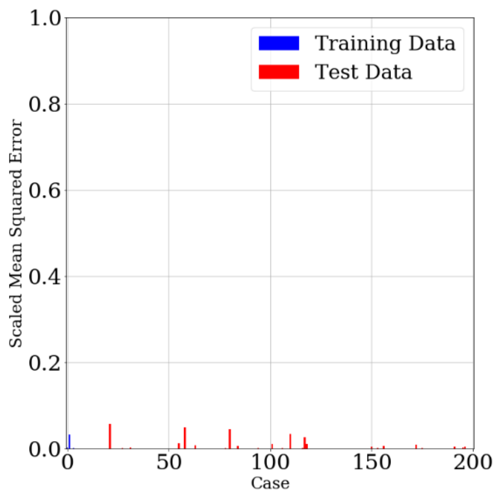


*Figure 7.48 Mean Squared Error, 100 epochs*

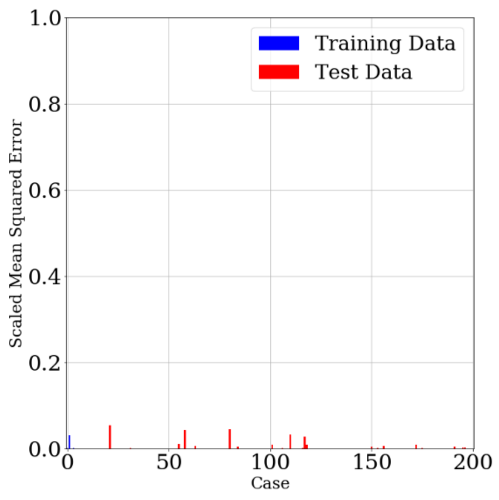


*Figure 7.49 Mean Squared Error, 1000 epochs*

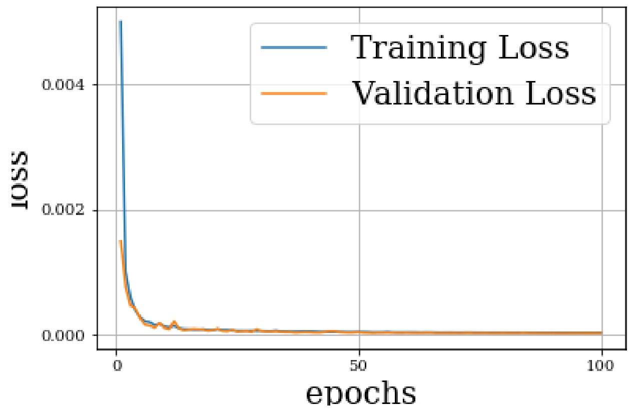


*Figure 7.50 Learning curves with 100 epochs*

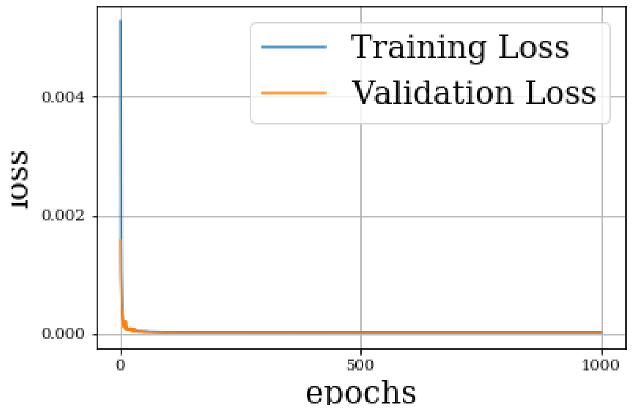


*Figure 7.51 Learning curves with 1000 epochs*

The final results with the GRU model are the following:

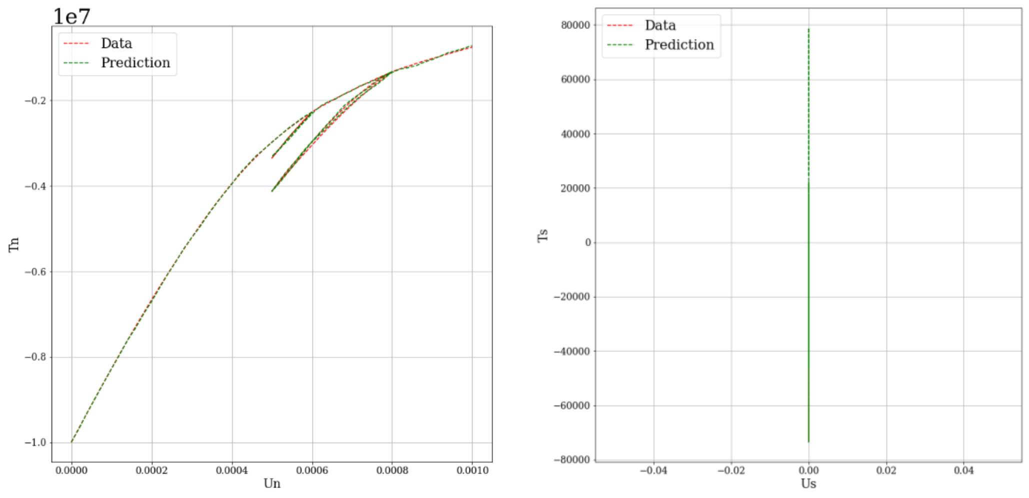


*Figure 7.52 Case 1*

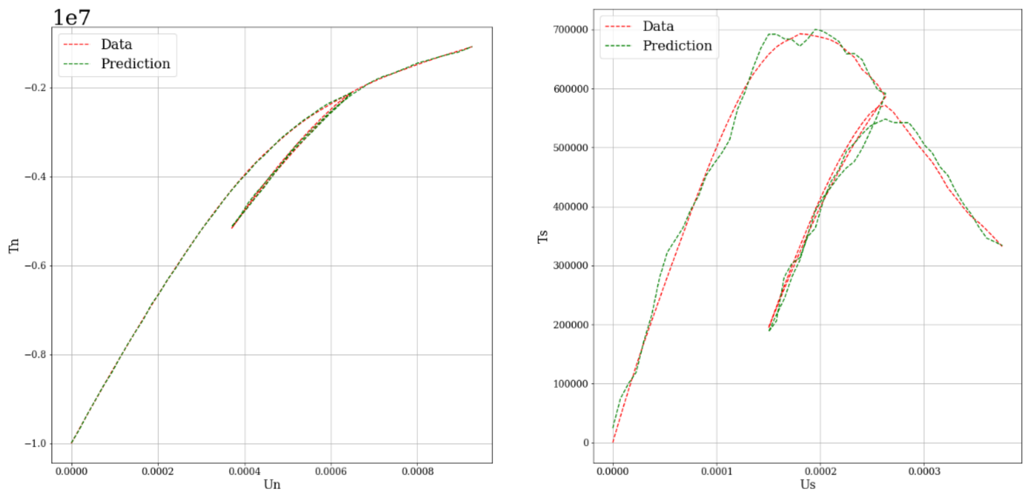


*Figure 7.53 Case 2*

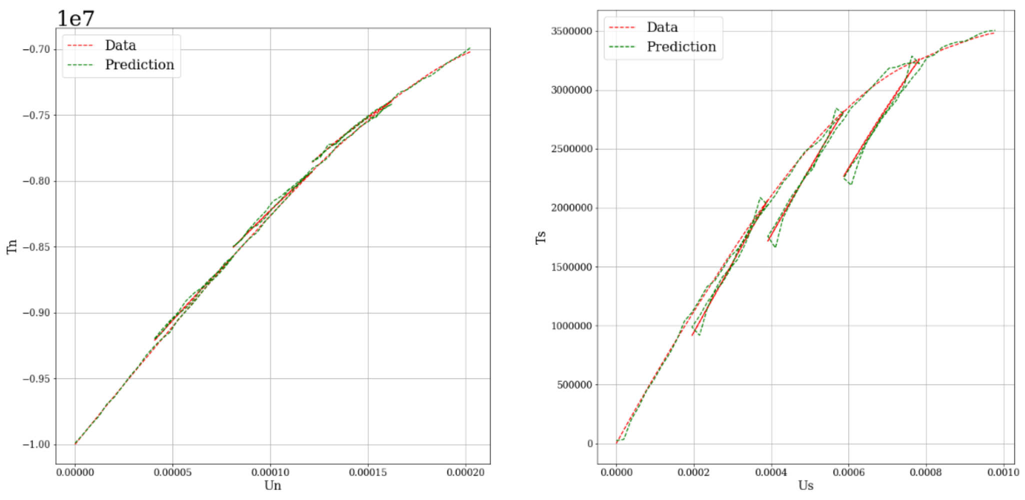


*Figure 7.54 Case 10*

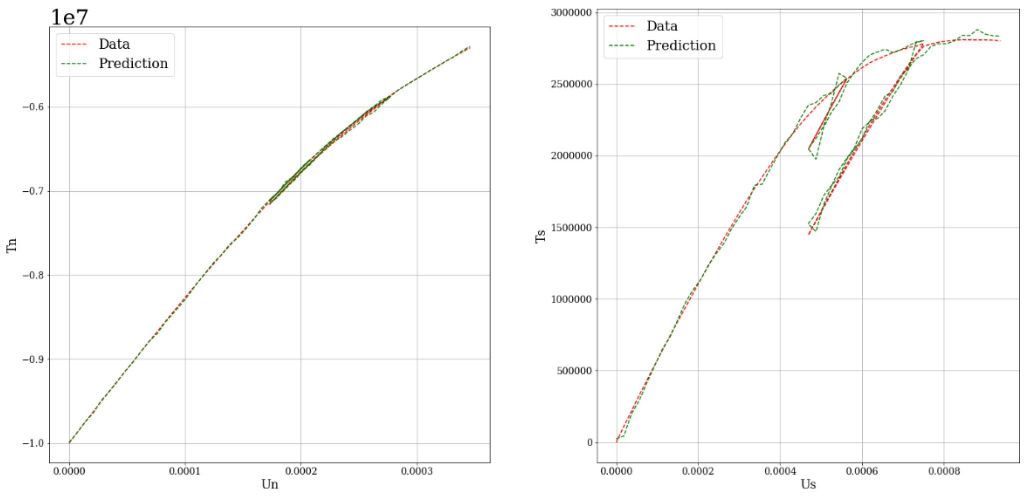


*Figure 7.55 Case 17*

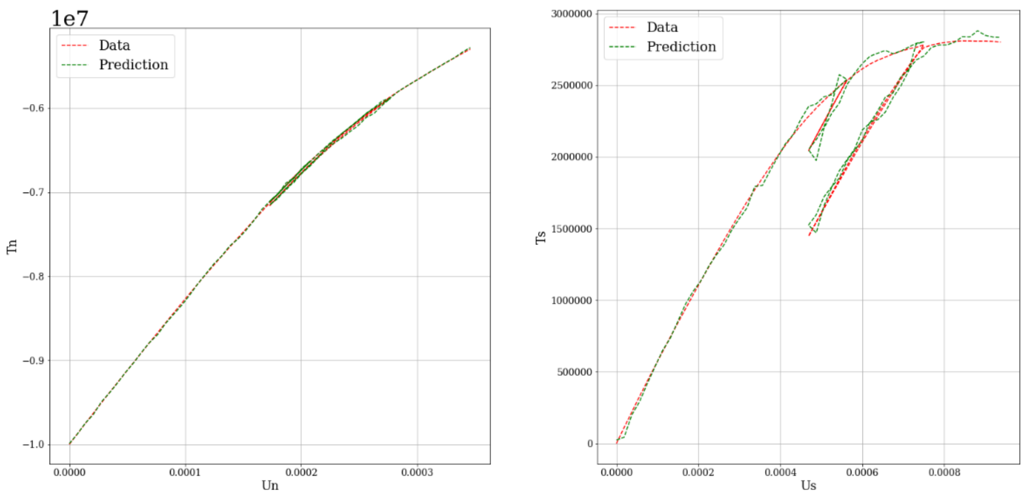


*Figure 7.56 Case 20*

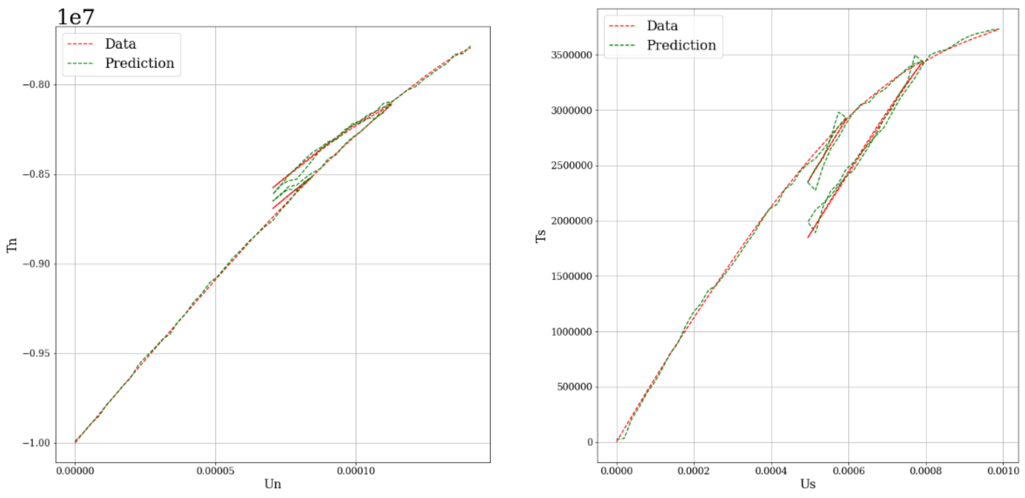


*Figure 7.57 Case 50*

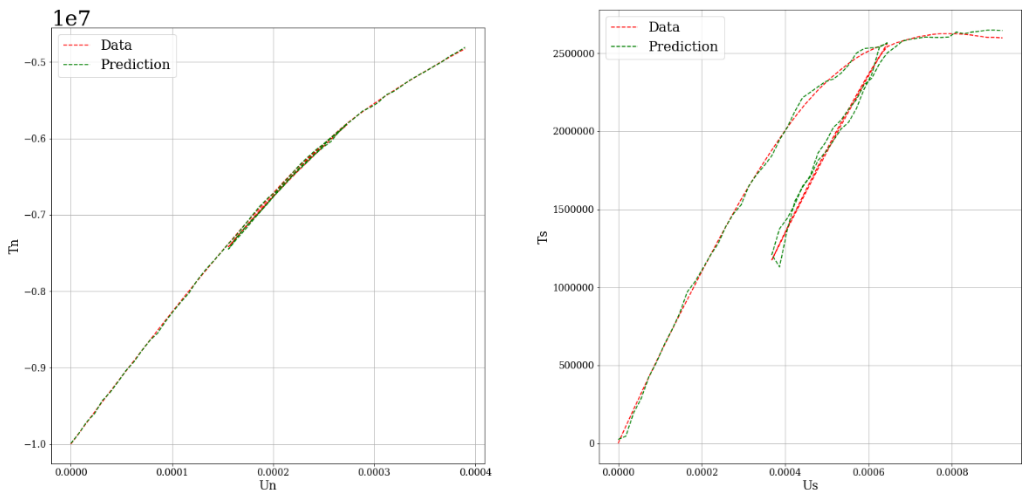


*Figure 7.58 Case 100*

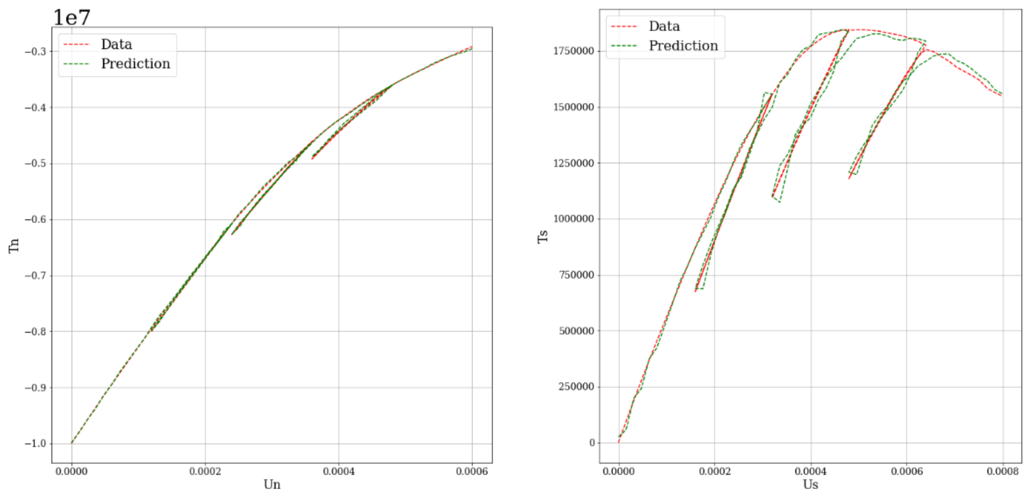
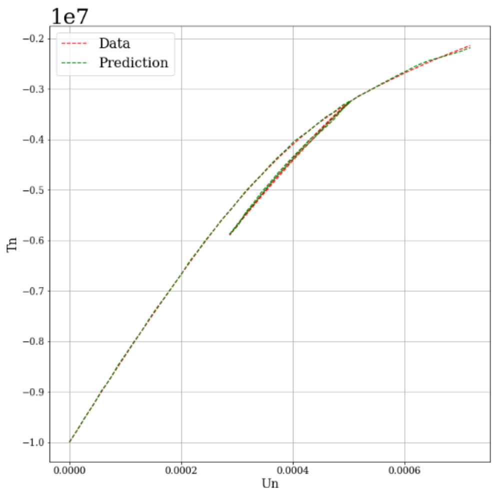


*Figure 7.59 Case 170*



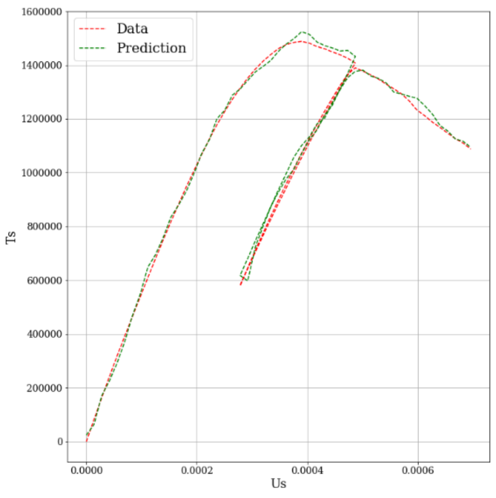


*Figure 7.60 Case 200*

# 8. Conclusions

Starting from the DEM simulations, in few hours a huge amount of data has been created instead of doing hundreds of laboratory experiments.

Furthermore, this machine learning approach helps us to create new and more complicated constitutive laws that do not require human supports just by coding a proper neural network.

In fact, the machine learning approach has been successful because it can predict more complicated traction-separation constitutive laws giving the displacement as input and the tractions as output. Due to the nature of granular materials. It has been possible to classify which property better represents the behaviour of the material. In particular, the Fabric Tensor that indicates how the single grains are oriented improves well the constitutive laws.

Second, the coordination number that characterizes how connected are the grains and last the porosity which is a macroscopic feature.

The intrinsic nature of the algorithms used in this work have been displayed. In particular, the differences between a simple layer and a recurrent neural network with memory cells.

This method allows to predict data with a black box model, which is not easy to comprehend. The mechanism and the single decisions made by the network in each hidden layer lead to a sort of knowledge of the algorithm hard to comprehend.

For sure this approach is able to best predict any kind of materials by just adjusting the hyperparameters of the networks because the algorithm generates relationship among different measurable physical quantities.

New development of this work can be done by adding algorithms that automatically choose the best value of all the hyperparameters so that the human error could be minimized.

# Bibliography

- J. Planas, B.Sanz, J.M. Sancho, (2019). A *first approach to comparing cohesive traction-separation laws for concrete.* 10th International Conference on Fracture Mechanics of Concrete and Concrete Structures, FraMCoS-X.
- K. Wang, W.C. Sun, (2019). *Meta-modeling game for deriving theory-consistent, micro-structure-based traction-separation laws via deep reinforcement learning,* Computer Methods in Applied Mechanics and Engineering.
- Y. Heider, K. Wang, W.C. Sun, (2019), *SO(3)-invariance of graph-based deep neural network for anisotropic elastoplastic materials,* Computer Methods in Applied Mechanics and Engineering.
- K. Wang, (2019). *From multiscale modeling to metamodeling of geomechanics problems, Columbia University.*
- K. Wang, W.C. Sun, (2018), *A multiscale multi-permeability poroplasticity model linked by recursive homogenizations and deep learning*, Computer Methods in Applied Mechanics and Engineering.
- E. C. Bryant, W. Sun, (2018). *A mixed-mode phase field fracture model in anisotropic rocks with consistent kinematics*. Computer Methods in Applied Mechanics and Engineering.
- Chen, Daniel Y, (2018). *Pandas for Everyone: Python Data Analysis*. Boston
- T. Kirchdoerfer, M. Ortiz, (2017). *Data driven computing with noisy material data sets*. ArXiv preprint arXiv:1702.01574.
- McKinney, Wes (2017). *Python for Data Analysis : Data Wrangling with Pandas, NumPy, and IPython* (2nd ed.).
- T. Kirchdoerfer and M. Ortiz, (2016). *Data-driven computational mechanics*. Computer Methods in Applied Mechanics and Engineering, 304:81–101.
- Guttag, John V., (2016). *Introduction to Computation and Programming Using Python: With Application to Understanding Data*. MIT Press.
- M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al., (2016). *Tensorflow: Large-scale machine learning on heterogeneous distributed systems*. arXiv preprint arXiv:1603.04467.
- F. Chollet et al. (2015). *Keras*. https://github.com/fchollet/keras.
- D. P. Kingma and J. Ba, (2014). Adam: *A method for stochastic optimization*. arXiv preprint arXiv:1412.6980.
- Cho et al., (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*.
- E. Bressert (2012). *SciPy and NumPy: an overview for developers.*
- Kyoungsoo Park, Glaucio H. Paulino, (2011). *Cohesive zone models: a critical review of traction-separation relationships across fracture surfaces.*

- K. Park and G. H. Paulino, (2011). *Cohesive zone models: a critical review of traction-separation relationships across fracture surfaces*. Applied Mechanics Reviews, 64(6):060802.
- Millman, K. Jarrod; Aivazis, Michael (2011). *Python for Scientists and Engineers*. Computing in Science and Engineering.
- F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. (2011). *Scikit-learn: Machine learning in Python*. Journal of Machine Learning Research, 12:2825–2830.
- V. ˇSmilauer, E. Catalano, B. Chareyre, S. Dorofeenko, J. Duriez, A. Gladky, J. Kozicki, C. Modenese, L. Scholt`es, L. Sibille, et al.(2010) *Yade documentation*. The YadeProject. http://yade-dem. org/doc/.
- Hastie, Tibshirani, Friedman, (2009). *The element of statistical learning,* Springer, New York.
- Freed, Y., and Banks-Sills, L., (2008). *A New Cohesive Zone Model for Mixed Mode Interface Fracture in Bimaterials*. Eng. Fract. Mech., 75(15), pp. 4583–4593.
- C. Bishop, (2006). *Pattern recognition and machine learning,* Springer, New York.
- Aydin, R. I. Borja, P. Eichhubl, (2006). *Geological and mathematical framework for failure modes in granular rock*. Journal of Structural Geology, 28(1):83–98, 2006.
- Kandula, S. S. V., Abanto-Bueno, J., Geubelle, P. H., and Lambros, J., (2005). *Cohesive Modeling of Dynamic Fracture in Functionally Graded Materials*. Int. J. Fract.
- T.L. Anderson, (2005). *Fracture Mechanics: Fundamentals and Applications.* CRC Press.
- Tan, H., Liu, C., Huang, Y., and Geubelle, P. H., (2005). *The Cohesive Law for the Particle/Matrix Interfaces in High Explosives.* J. Mech. Phys. Solids.7.
- Scheider, W. Brocks, (2003). *The Effect of the Traction Separation Law on the Results of Cohesive Zone Crack Propagation Analyses*. Key Engineering Materials 251–252,313–318. https://doi.org/10.4028/www.scientific.net/kem.251-252.313
- M. Elices, G. Guinea, J. Gomez, and J. Planas, (2002). *The cohesive zone model: advantages, limitations and challenges*. Engineering fracture mechanics, 69(2):137–163.
- Ortiz, M., and Pandolfi, A., (1999). *Finite-Deformation Irreversible Cohesive Elements for Three Dimensional Crack-Propagation Analysis*. Int. J. Numer. Methods Eng.
- Geubelle, P. H., and Baylor, J. S., (1998). *Impact-Induced Delamination of Composites: A 2D Simulation*. Compos. Part B: Eng.

- S. Hochreiter and J. Schmidhuber,(1997). *Long short-term memory. Neural computation*, 9(8):1735–1780.
- K. Bagi, (1996). *Stress and strain in granular assemblies. Mechanics of materials*, 22(3):165–177.
- T. L. Anderson, (1995). *Fracture Mechanics – Fundamentals and applications Second Edition*, CRC Press.
- C. Bishop, (1995). *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford.
- Tvergaard, V., and Hutchinson, J. W. (1993). *The Influence of Plasticity on Mixed Mode Interface Toughness*. J. Mech. Phys. Solids.
- Xu, X. P., and Needleman, A., (1993). *Void Nucleation by Inclusion Debonding in a Crystal Matrix.* Model. Simul. Mater. Sci. Eng., 1(2), pp. 111–132.
- G. Van Rossum, (1993). *An Introduction to Python for UNIX/C Programmers*. Proceedings of the NLUUG Najaarsconferentie (Dutch UNIX Users Group).
- Rice, J. R., (1992). *Dislocation Nucleation From a Crack Tip: An Analysis Based on the Peierls Concept*. J. Mech. Phys. Solids, 40(2), pp. 239–271.
- Beltz, G. E., and Rice, J. R., (1991). *Dislocation Nucleation Versus Cleavage Decohesion at Crack Tips. Modeling the Deformation of Crystalline Solids Presented.*
- Tvergaard, V., (1990). *Effect of Fibre Debonding in a Whisker-Reinforced Metal*. Mater. Sci. Eng., A125(2), pp. 203–213.
- Needleman, A., (1987). *A Continuum Model for Void Nucleation by Inclusion Debonding*. ASME J. Appl. Mech., 54(3), pp. 525–531.
- Satake, (1982). *Fabric tensor in granular materials*. In IUTAM Conference on Deformation and failure of Granular Materials.
- J. Christoffersen, M. Mehrabadi, S. Nemat-Nasser, (1981). *A micromechanical description of granular material behavior*. Journal of Applied Mechanics, 48(2):339–344.
- Rose, J. H., Ferrante, J., and Smith, J. R., 1981, “Universal Binding Energy Curves for Metals and Bimetallic Interfaces,” Phys. Rev. Lett., 47(9), pp. 675–678
- P. A. Cundall and O. D. Strack (1979). *A discrete numerical model for granular assemblies*. Geotechnique, 29(1):47–65.
- G.I. Barenblatt, (1962). *The mathematical theory of equilibrium cracks in brittle fracture. Advances in Applied Mechanics.* pp. 55–129.
- Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. IBM Journal of research and development, 3(3), 210-229.

- Peierls, R., 1940, “The Size of a Dislocation,” Proc. Phys. Soc., 52(1), pp. 34–37.
- https://machinelearningmastery.com/
- https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense
- Machine learning course on YouTube, Caltech University, https://www.youtube.com/watch?v=mbyG85GZ0PI&t=3678s