PMR5251 - Assessment of Mechanical Behavior of Materials using Machine Learning Approach

UNIVERSIDADE DE SÃO PAULO • ESCOLA POLITÉCNICA •

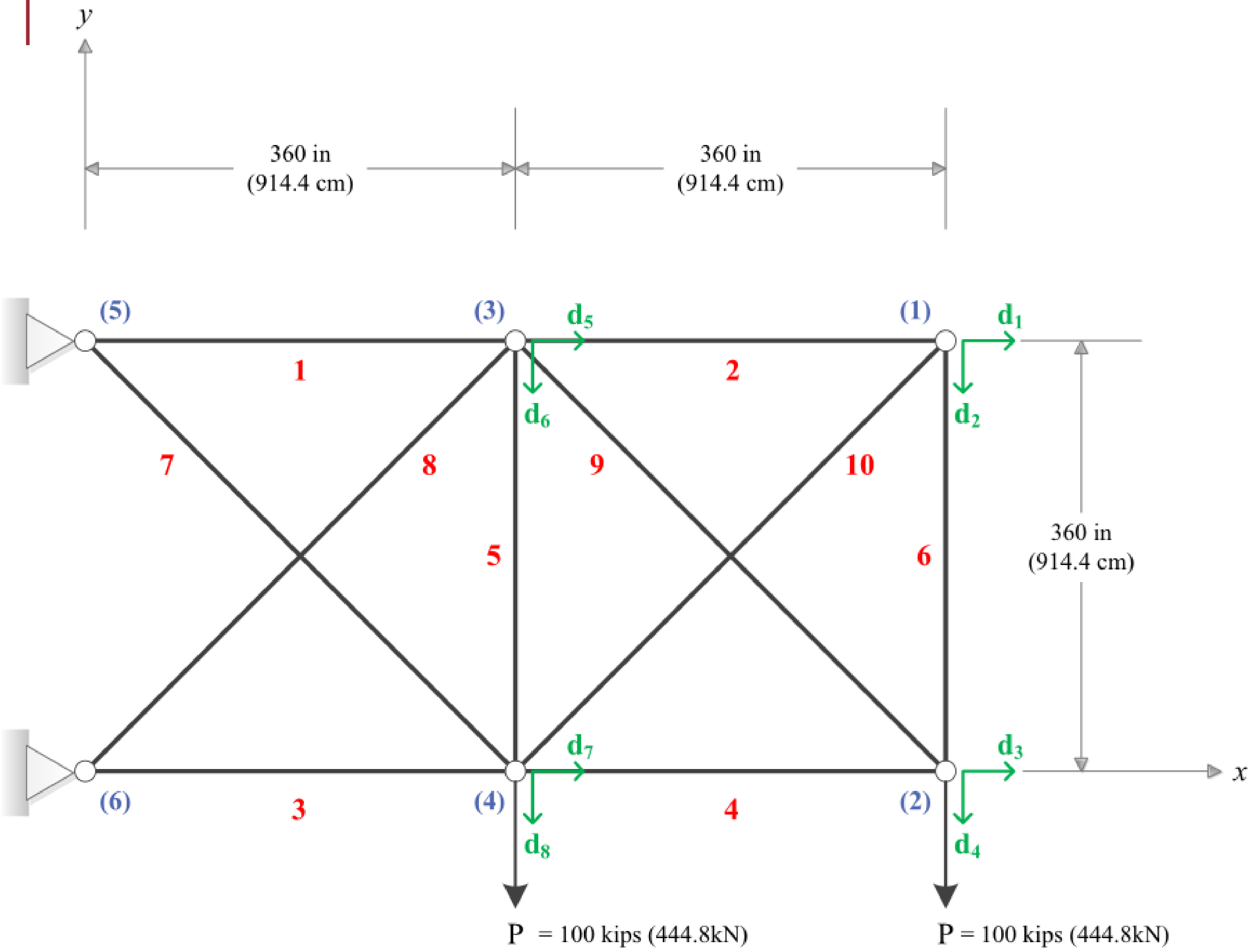
ARTIFICIAL NEURAL NETWORKS (ANNS)

Izabel F. Machado
Larissa Driemeier

OUR PROBLEM

Structural bars

10-BARS TRUSS STRUCTURE



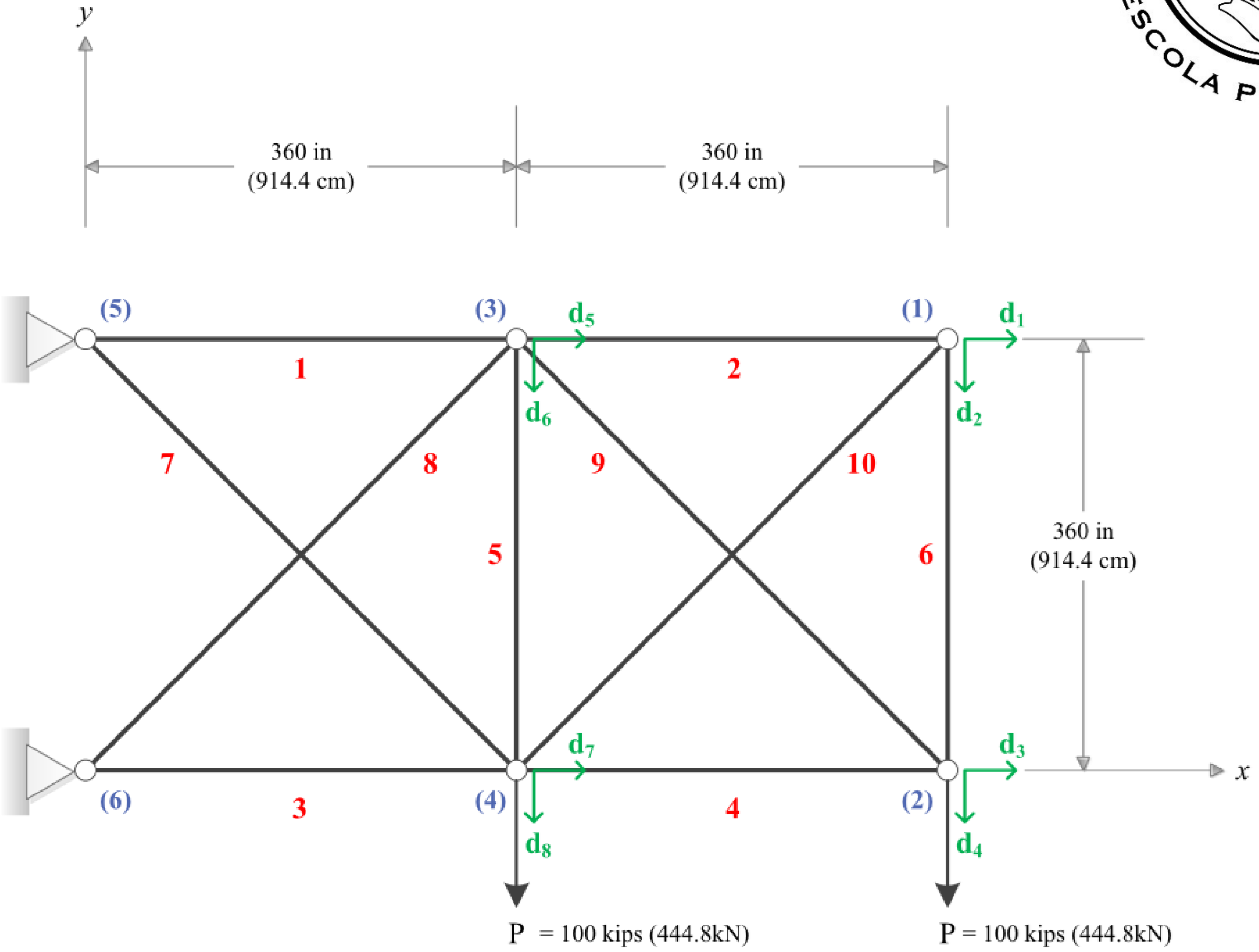
Property	Value
Density $[ton/mm^3]$	$2.768\ 10^{-9}$
Poisson $[-]$	0.35
Young Modulus $[MPa]$	68950

Seunghye Lee, Jingwan Ha, Mehriniso Zokhirova, Hyeonjoon Moon, **Background Information of Deep Learning for Structural Engineering**, July 2017, *Archives of Computational Methods in Engineering*

INPUT DATA GENERATION

Cross sectional areas randomly choosen.

Values varying from $0.6\ cm^2$ to $225.8\ cm^2$



AREA GENERATION

The Python script for generating areas is called **gera_areas_10.py and it is available in the Moodle.**

In the script 520 datasets are generated, with 10 random areas each, using the command:

```
num = random.random()*(225.8-.6)+0.6
```

The data is written to a **csv** file, which will be imported by Notebook.

However, if you do not want to generate the areas with the code in Python, the file is already available in Moodle with the name **areas.csv**

OUTPUT DATA GENERATION

To generate the output data, you need the following files:

1. areas.csv
2. 10-BarStructure.py
3. BasicInput.inp

The **file 1** contain 520 area combinations.

The **file 2** is the script in Pyhton used to run Abaqus.

The **file 3** is a *template* Abaqus file, which contains geometry, material and loading data. The areas will be modified by the script (file 2), which will also do the analysis and store the results.

	A	[illegible]	[illegible]	D	E	F	G	H	I	J	K	L	M	N
1	iteration	area1	[illegible]	area3	area4	area5	area6	area7	area8	area9	area10	d1	d2	d3
2	0	[illegible]	[illegible]	0.0172602044194	0.00580415445965	0.0112171981613	0.010182538779	0.0147338737457	0.0178220498676	0.00217371789416	0.000698385171276	105.791.368.484	-70.774.597.168	-243.559.417.725
3	1	[illegible]	[illegible]	0.017226547457	0.000107428321467	0.0100901196101	0.0163090815283	0.00521172522301	0.0213474960639	0.0203601463454	0.000748886417916	936.634.063.721	-245.959.854.126	-176.197.128.296
4	2	[illegible]	[illegible]	0.0212096391458	0.00864471943274	0.00493781842338	0.00956606528212	0.000713998536186	0.00505249632447	0.00992122860901	0.0112256916759	412.073.554.993	-185.375.213.623	-96.529.586.792
5	3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.0125913513461	0.0145244690532	0.00424660910795	95.518.579.483	-698.960.494.995	-151.715.927.124
6	4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.00956584963913	0.0187524038125	0.0151552813556	962.206.172.943	-464.393.310.547	-871.774.101.257
7	5	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.00552650420419	0.0180175436549	0.00939035126425	344.701.499.939	-1.566.275.177	-661.502.914.429
8	6	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.0175905276898	0.0117915331646	0.00891610473859	908.464.336.395	-480.551.185.608	-185.923.099.518
9	7	0.0110878980808	0.000726028188535	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.00389626391319	0.0113704123359	0.0221763658774	343.373.603.821	-264.447.845.459	-154.565.086.365
10	8	0.017412181109	0.0122121849391	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.0103996466026	0.00612417383198	0.0124008768892	166.513.576.508	-546.397.094.727	-104.529.037.476
11	9	0.0216142586585	0.000188569595223	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.0117406349449	0.0127017791148	0.00965556210658	158.445.119.858	-602.686.729.431	-551.290.369.034
12	10	0.00132389666017	0.0196526286946	0.0128963849989	0.00456038374239	0.0114263049265	0.0109805135274	0.00809491000155	0.00785367473629	0.01218654248	0.014100982477	170.779.399.872	-686.803.588.867	-268.583.431.244
13	11	0.0138524295069	0.0103774659382	0.000689996641568	0.00523070530436	0.0040507975513	0.0132220588099	0.0194499195464	0.0180408449418	0.0180106371106	0.018446169585	569.292.831.421	-58.118.221.283	-103.867.931.366
14	12	0.00580922178277	0.0190160936228	0.0152185165929	0.00193443278334	0.000435872990202	0.000387890635307	0.0170758141787	0.00568007376172	0.00252568388667	0.0141305429351	253.977.928.162	-585.195.655.823	-14.636.384.964
15	13	0.00781640289945	0.00162548632451	0.00365476681611	0.0119366065866	0.00384662418893	0.00620603311716	0.0160850051602	0.0102998807086	0.00731147977904	0.0107293232391	153.299.427.032	-103.677.566.528	-117.646.932.602
16	14	0.000592250688272	0.00876526599923	0.00953908865579	0.004294645143	0.00250931331387	0.020323912628	0.0115478118905	0.00476872915227	0.0136992073736	0.0184597333319	247.895.946.503	-791.557.922.363	-33.369.644.165
17	15	0.000528823803629	0.000462309009042	0.00335831839379	0.0162481748466	0.00366832538602	0.0159277187392	0.0153325189096	0.0123266927238	0.00502790632547	0.0220303885413	217.263.355.255	-136.403.640.747	-338.654.022.217
18	16	0.0180267005155	0.0116938211217	0.00508636897116	0.0146643645356	0.00895310318201	0.013028051082	0.00729445562645	0.0142689458358	0.00138384081697	0.00678460598551	146.168.737.411	-707.486.877.441	-203.440.341.949
19	17	0.0218571825446	0.0197770311802	0.0069598266899	0.0193937444311	0.00704938888799	0.0212127754917	0.0168113245124	0.00943219935747	0.00574310446334	0.000250975510682	976.124.763.489	-514.930.915.833	-113.129.272.461
20	18	0.0198487270677	0.00091388026908	0.018513205771	0.0217287693391	0.0129027184419	0.0039225649834	0.0196024295711	0.0219894183183	0.0159146011653	0.0115198367617	744.098.472.595	-463.107.032.776	-661.952.590.942
21	19	0.00857185814942	0.00787288352035	0.00469375477428	0.0152419258808	0.00981003672718	0.00443155188507	0.00241163349839	0.0150573635368	0.00672755659783	0.0113154942488	10.888.250.351	-116.728.744.507	-482.958.068.848
22	20	0.00738678414781	0.0196889163472	0.0203207546322	0.000467453991583	0.00458320981765	0.00744072067877	0.022283596477	0.0176864124614	0.0076964339896	0.0048574310145	123.831.996.918	-644.744.186.401	-71.123.380.661
23	21	0.0152487281702	0.0189250280999	0.021052861867	0.00780349782909	0.0199314949195	0.0155337213021	0.0109709112332	0.0222536453357	0.0053441025933	0.0163974759942	108.373.708.725	-427.304.649.353	-11.968.003.273
24	22	0.00196699878898	0.00388151207321	0.0205754448846	0.00485604375121	0.0171552964348	0.0135767028546	0.0190022970473	0.00834979214662	0.00772322349224	0.0066181682725	21.316.526.413	-735.191.955.566	-171.238.918.304
25	23	0.0195942944272	0.0136616865506	0.0215510039355	0.0200412101582	0.00310799141095	0.012472359076	0.00240827295529	0.000941383224402	0.0017083157921	0.0195661114079	909.415.130.615	-242.506.881.714	-673.733.444.214

This step is finished.

It was your homework ...

File **FinalResult.csv**

We are ready to go to the Notebook to work with the neural network, as we have:

Input files – areas.csv, which has all combinations of areas

Output file – FinalResult.csv, with areas (yes, again...), displacements of all nodes and stresses of all bars.

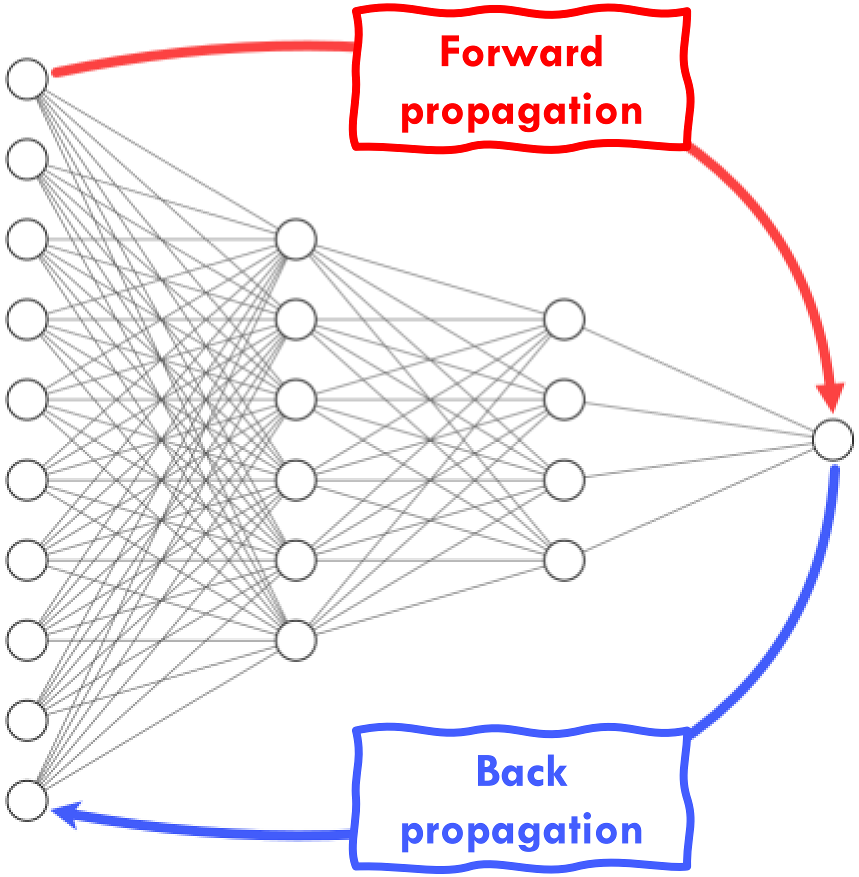
ANN

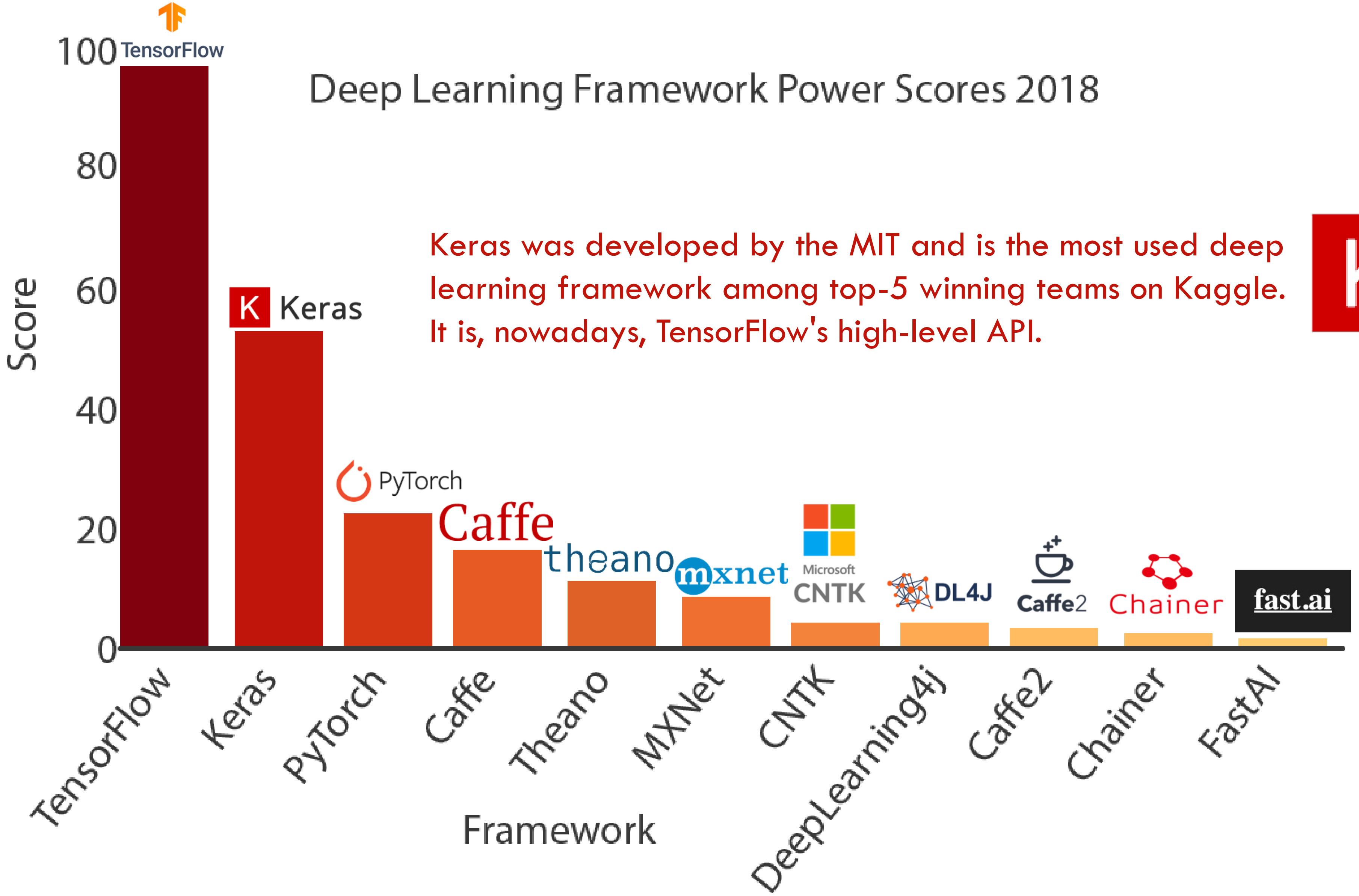
Machine learning tools

REDE NEURAL ARTIFICIAL

Virtually no one develops its own code to implement and train an ANN since there are numerous development tools, already tested, that do most of this work and are widely used.

The great advantage of using one of these tools comes from the fact that we only need to define the configuration (architecture) of the ANN, that is, to define how **forward propagation** is performed. When forward propagation is defined, the **back propagation**, which is in fact the most difficult part of coding an ANN, is automatically generated using symbolic manipulation.





Keras was developed by the MIT and is the most used deep learning framework among top-5 winning teams on Kaggle. It is, nowadays, TensorFlow's high-level API.

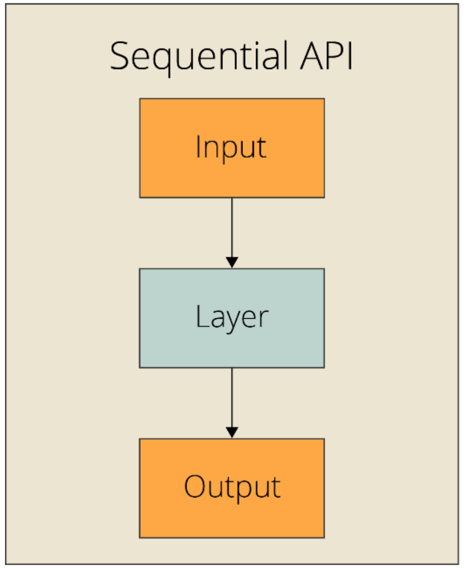
NOTEBOOK

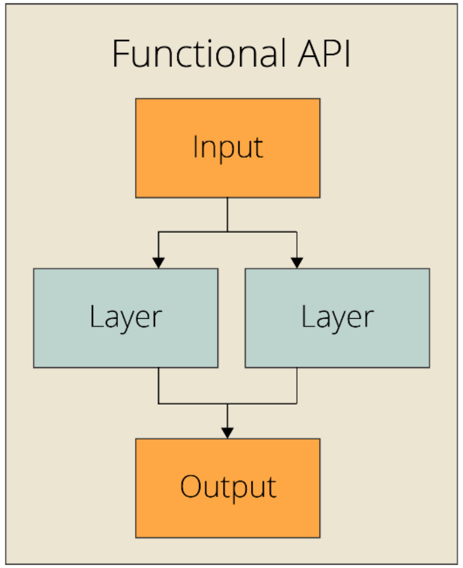
PMR5251_C03_2023.ipynb

KERAS

```
import tensorflow as tf
from tensorflow import keras
```

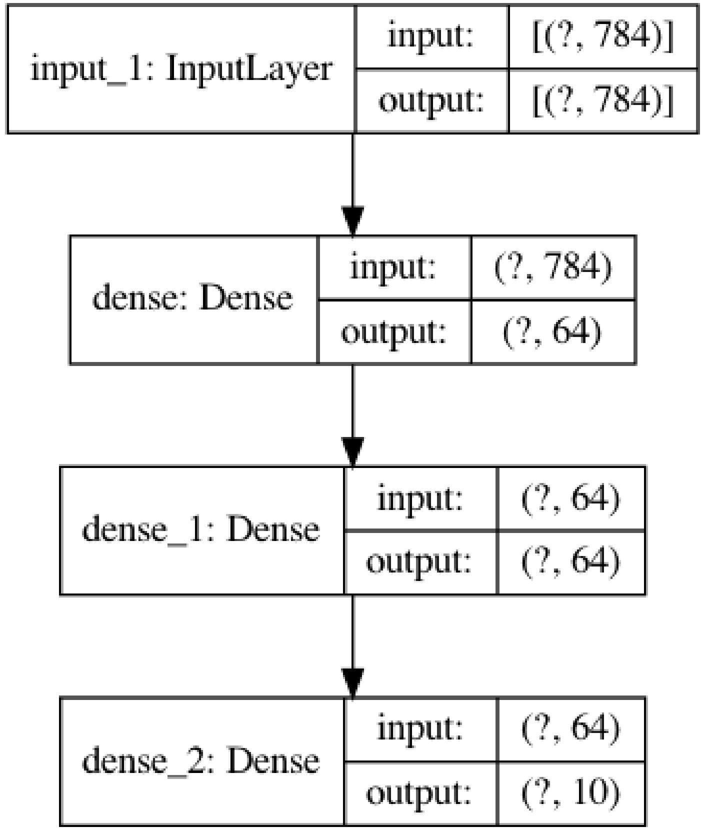
In Keras there are two ways to define an ANN…





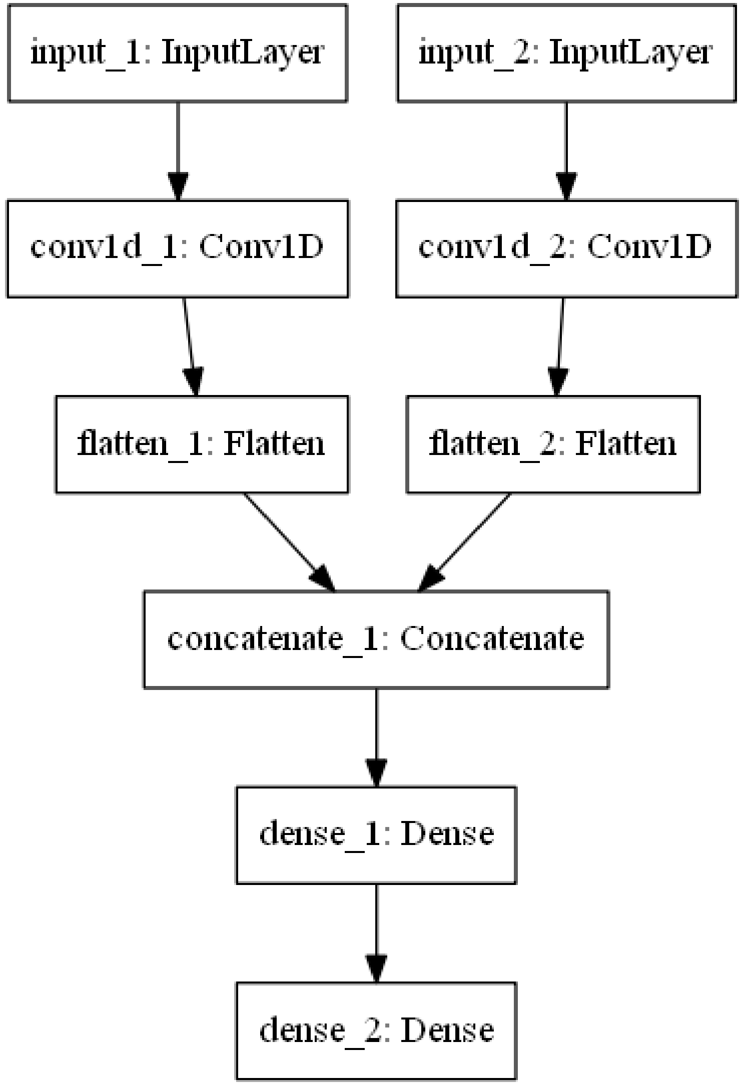
SEQUENTIAL API MODE

It is the simplest model and it comprises a linear pile of layers that allows you to configure models layer-by-layer for most problems. The sequential model is very simple to use, however, it is limited in its topology. The limitation comes from the fact that you are not able to configure models with shared layers or have multiple inputs or outputs.

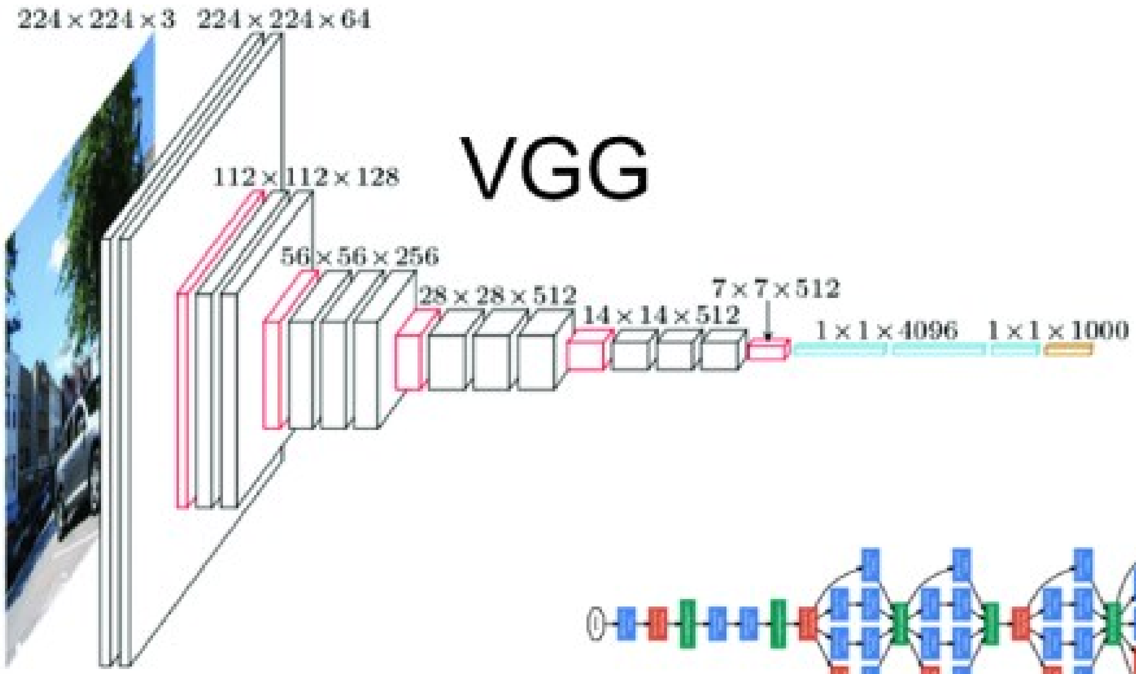


FUNCTIONAL API MODE

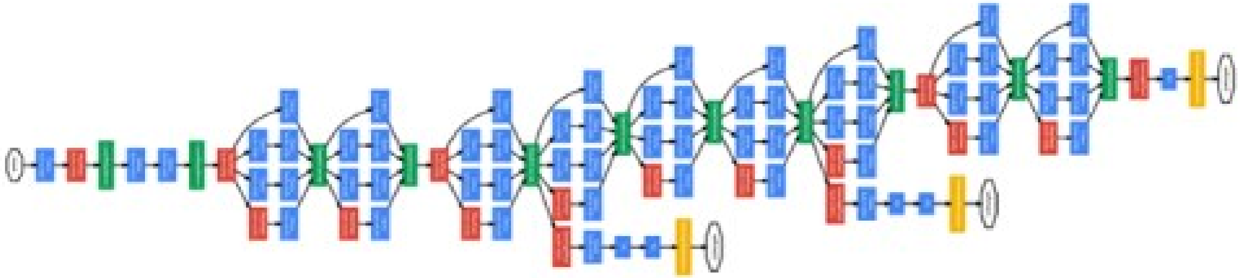
It is ideal for creating complex models, that require extended flexibility. It allows you to define models that feature layers connect to more than just the previous and next layers. With this model becomes possible to create complex networks such as siamese networks, residual networks, multi-input/multi-output models and models with shared layers.



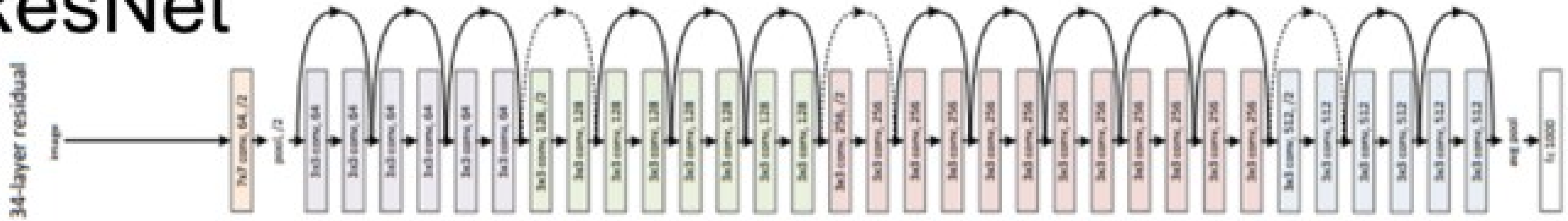
VGG



GoogLeNet



ResNet



OUR JOB TODAY...

IMPORT LIBRARIES

```
import tensorflow as tf
from tensorflow import keras
```

Throughout this notebook the new version 2.12 of Tersorflow was used, with built-in keras support, which has been recently released to the public.

```
import numpy as np
import pandas as pd
```

Pandas is the most widely used open source Python package for data analysis and machine learning. It is built on top of another package called Numpy (see that it was imported before Pandas in our code), which provides support for matrix analysis.

UPLOAD YOUR FILE INITIAL **FINALRESULT.CSV**

The script in item 02 automatically generates the bar geometry in Abaqus. If you want to build up a geometry - at least once - with Abaqus, Prof Marcilio Alves kindly prepared a tutorial that can be accessed through the link.

```
from google.colab import files
uploaded = files.upload()
```

Escolher arquivos Nenhum arquivo selecionado Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving areas.csv to areas.csv
Saving FinalResult.csv to FinalResult.csv
```

As shown below, there are 10 different areas, which will be the inputs, and various other measurements, which might be used as outputs of the Neural Network

DATASET

```
df = pd.read_csv('FinalResult.csv', index_col=0)
train.head()
```

	area1	area2	area3	area4	area5	area6	area7	area8	area9	area10	d1	d2	d3
iteration													
434	0.014080	0.013758	0.010357	0.010827	0.019467	0.007538	0.014179	0.003238	0.022078	0.001040	26.458395	-86.712097	-14.344073
436	0.009107	0.019709	0.008960	0.000679	0.005468	0.019763	0.011006	0.008855	0.007129	0.006556	17.897791	-76.273315	-13.668117
208	0.008202	0.013466	0.014938	0.009267	0.017776	0.019281	0.006559	0.005118	0.009010	0.015793	25.154400	-82.572304	-22.442982
332	0.021670	0.012408	0.021819	0.022391	0.016985	0.012988	0.008441	0.001836	0.011957	0.004098	42.745960	-111.905495	-21.031446
220	0.010295	0.015570	0.011842	0.013228	0.007927	0.019004	0.005587	0.014462	0.009896	0.003410	10.272119	-66.871239	-23.419832

TRAIN AND TEST DATASET

SPLIT DATASET INTO TRAIN AND TEST

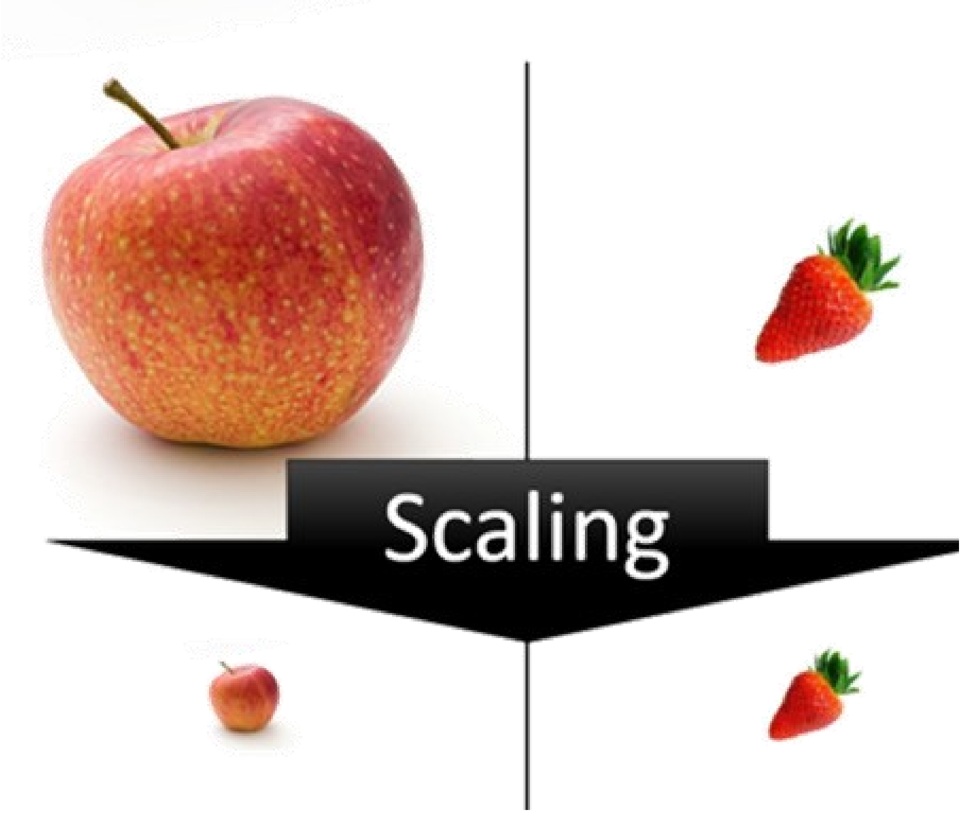
```
from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size=0.2, random_state=42)
```

```
x_train = train.loc[:,'area1':'area10'].values
y_train = train[['d4']].values
x_val = test.loc[:,'area1':'area10'].values
y_val = test[['d4']].values
print(x_train.shape, y_train.shape)
print(x_val.shape, y_val.shape)
```

```
(416, 10)
(416, 1)
(104, 10)
(104, 1)
```

SCALING

Most of times different features in the data might be have varying magnitudes. For example, a dataset containing two resources, displacement which ranges from 0-1) and stresses, about 100-1000 times greater than displacement. So, these two features are at very different ranges with high values dominating those with small values. The reason is that many of the machine learning algorithms use euclidean distance between data point in their computation. In this case, machine learning model treats those with small values as if they don't exist.



SCALING

NORMALIZATION VS STANDARDIZATION

Normalization

$$\bar{x}_i = \frac{x_i - \min \boldsymbol{x}}{\max \boldsymbol{x} - \min \boldsymbol{x}}$$

Also known as min-max scaling or min-max normalization, it is the simplest method and consists of rescaling the range of features to scale the range in [0, 1].

Normalization is good to use when the distribution of data does not follow a Gaussian distribution.

Standardization

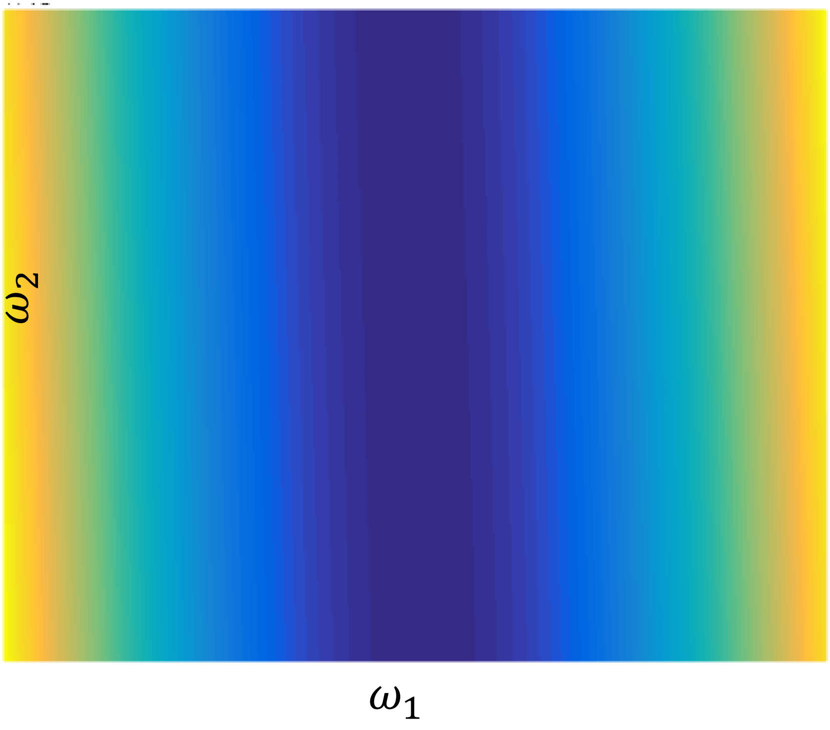
$$\bar{x}_i = \frac{x_i - \mu^{(i)}}{\sigma^{(i)}}$$

Feature standardization makes the values of each feature in the data have zero mean and unit variance.

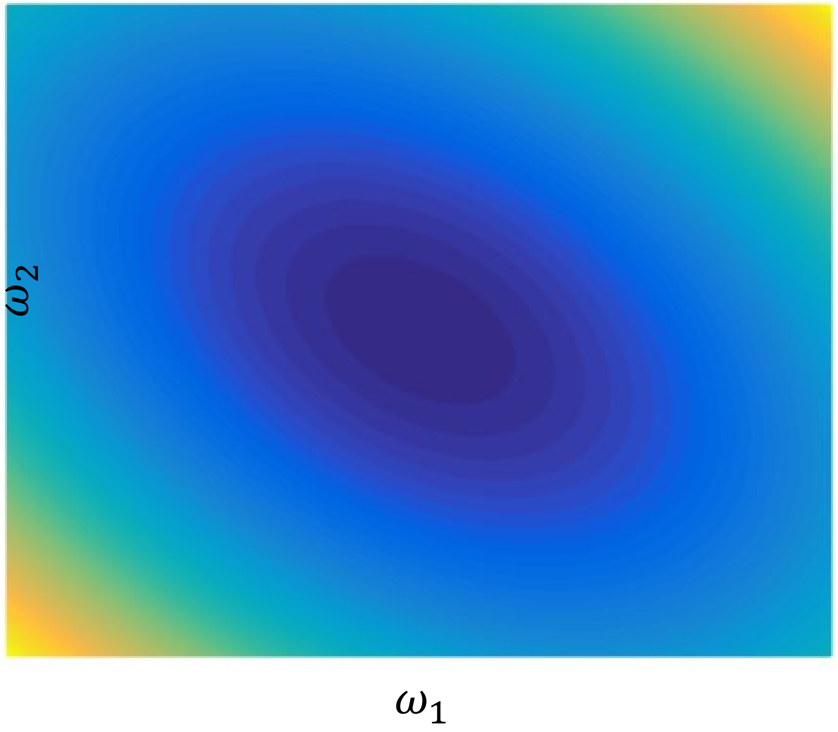
Standardization can be helpful in cases where the data follows a Gaussian distribution.

COST FUNCTION

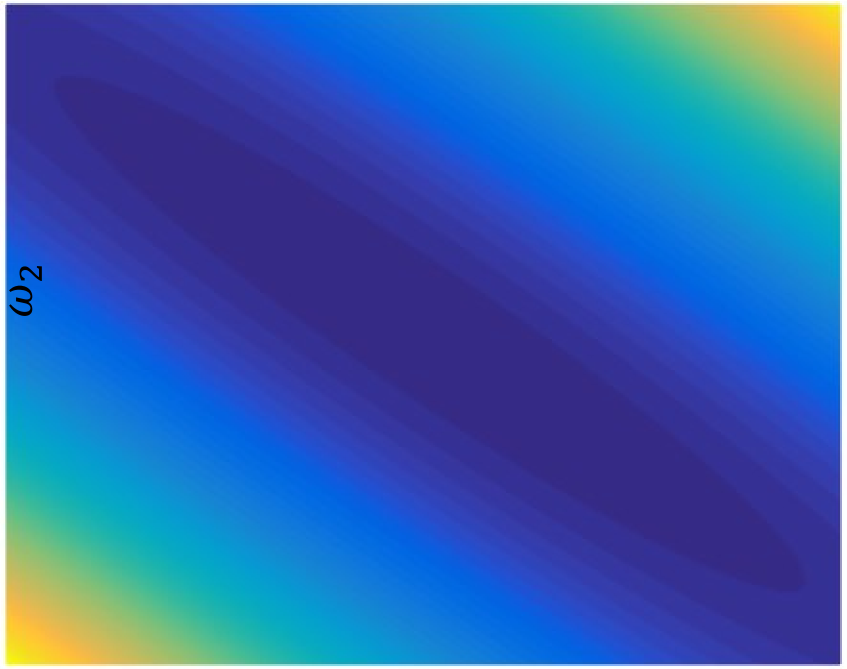
Raw data $x_1 \gg x_2$



Normalization



Standardization



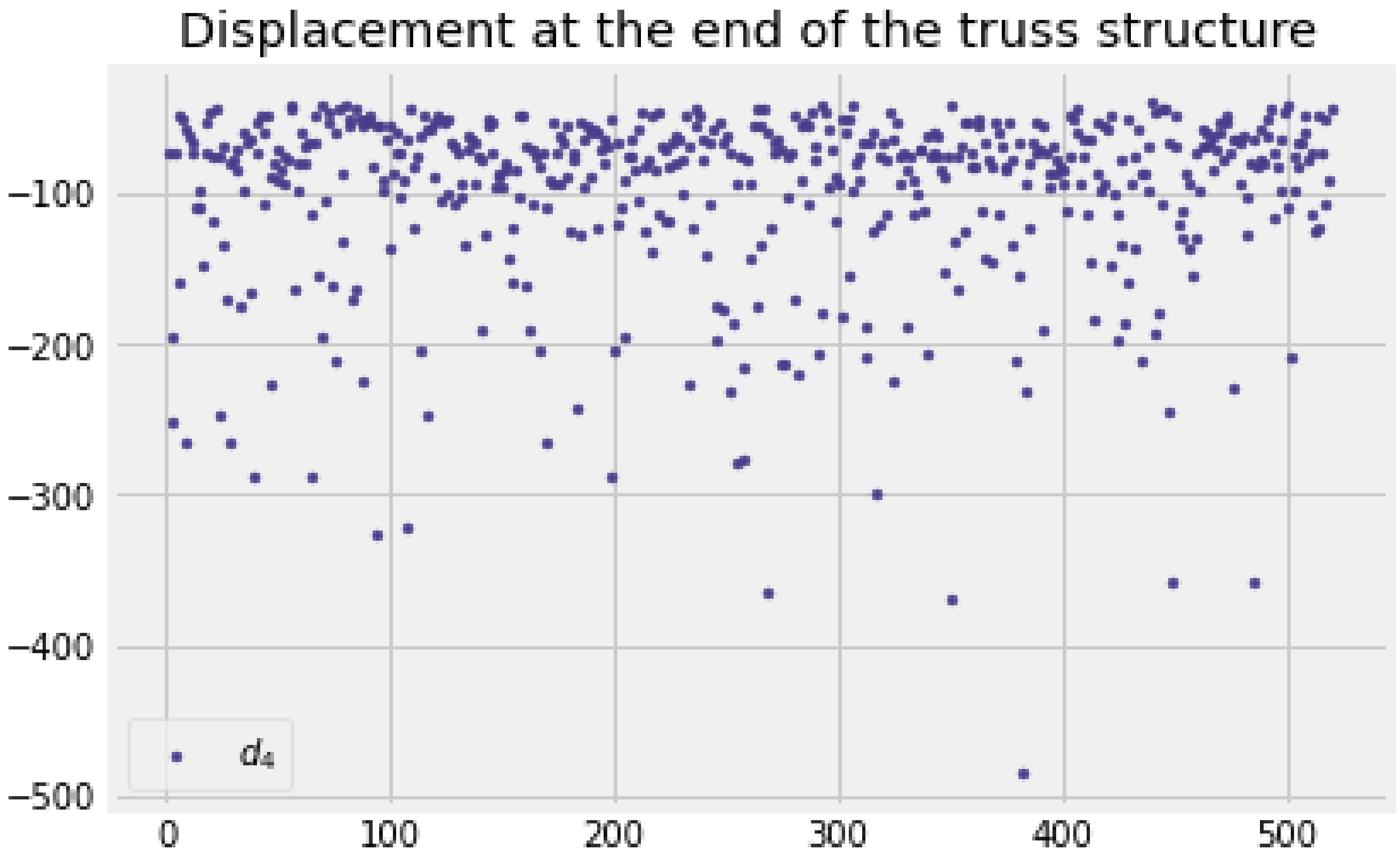
NORMALIZING DATASET

```
from sklearn.preprocessing import MinMaxScaler
# Scaling the input data using the MinMaxScaler from scikit-learn
scaler_x = MinMaxScaler().fit(x_train)
x_train_sca = scaler_x.transform(x_train)
x_val_sca = scaler_x.transform(x_val)
# Normalizing the output data using the normalizer from scikit-learn
normalizer_y = MinMaxScaler(feature_range = (-1.,0.)).fit(y_train)#StandardScaler,MaxAbsScaler
y_train_sca = normalizer_y.transform(y_train)
y_val_sca = normalizer_y.transform(y_val)
#  Min and Max in input
min_x_train = np.min(x_train_sca)
min_x_val = np.min(x_val_sca)
max_x_train = np.max(x_train_sca)
max_x_val = np.max(x_val_sca)
# Mean and Standard Deviation in Output
min_y_train = np.min(y_train_sca)#mean
min_y_val = np.min(y_val_sca)
max_y_train = np.max(y_train_sca)#std
max_y_val = np.max(y_val_sca)
```

SCALING RESULT

```
print(f'For the input training set, the min is {min_x_train} and the max is {max_x_train}')
print(f'For the input validation set, the min is {min_x_val} and the max is {max_x_val}')
print(f'For the output train set, the min is {min_y_train} and the max is {max_y_train}')
print(f'For the output validation set, the min is {min_y_val} and the max is {max_y_val}')
```

```
For the input training set, the min is 0.0 and the max is 1.0
For the input validation set, the min is -0.0023858525432659487 and the max is 1.0081953276506366
For the output train set, the min is -1.0 and the max is 5.551115123125783e-17
For the output validation set, the min is -1.3732910962953015 and the max is -0.006101485466849105
```


Displacement at the end of the truss structure
−100
−200
−300
−400
−500
0
100
200
300
400
500
d_4

OUR SEQUENCIAL NN WITH KERAS

We will start with the simplest way to create an RNA in Keras, which is the **sequential model**. Creating, training and testing an ANN with Keras is done in the following steps:

I. Definition of training and test data;
II. ANN configuration, which consists of defining the layers to map the inputs to the desired outputs;
III. Compilation of the ANN, which also includes configuring the training process by choosing the cost function, the optimizer and the metric to evaluate performance;
IV. ANN training;
V. ANN performance evaluation.

FIRST NEURAL NETWORK MODEL

```
from keras import models
from keras.layers import Dense, Activation

##First definition

model = models.Sequential([
    Dense(20, input_shape=(10,)),
    Activation('sigmoid'),
    Dense(1)
])

model.summary()
```

The hidden layer is of the dense type (fully connected), it has 20 neurons

its activation function is sigmoid

FIRST NEURAL NETWORK MODEL

```
from keras import models
from keras.layers import Dense, Activation

##First definition

model = models.Sequential([
    Dense(20, input_shape=(10,)),
    Activation('sigmoid'),
    Dense(1)
])

model.summary()
```

Input data for each training example is a 1-D vector (10)

the dimension of the second axis of the input tensor is not included in the 'input_shape' argument, because at that moment the number of examples that will be used in training is unknown

FIRST NEURAL NETWORK MODEL

```
from tensorflow.keras import models
from tensorflow.keras.layers import Dense, Activation

##First definition

model = models.Sequential([
    Dense(20, input_shape=(10,)),
    Activation('sigmoid'),
    Dense(1)
])

model.summary()
```

The hidden layer is of the dense type (fully connected), it has 20 neurons

its activation function is sigmoid

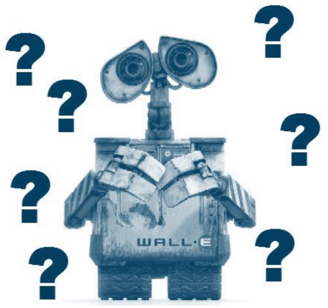
FIRST NEURAL NETWORK MODEL

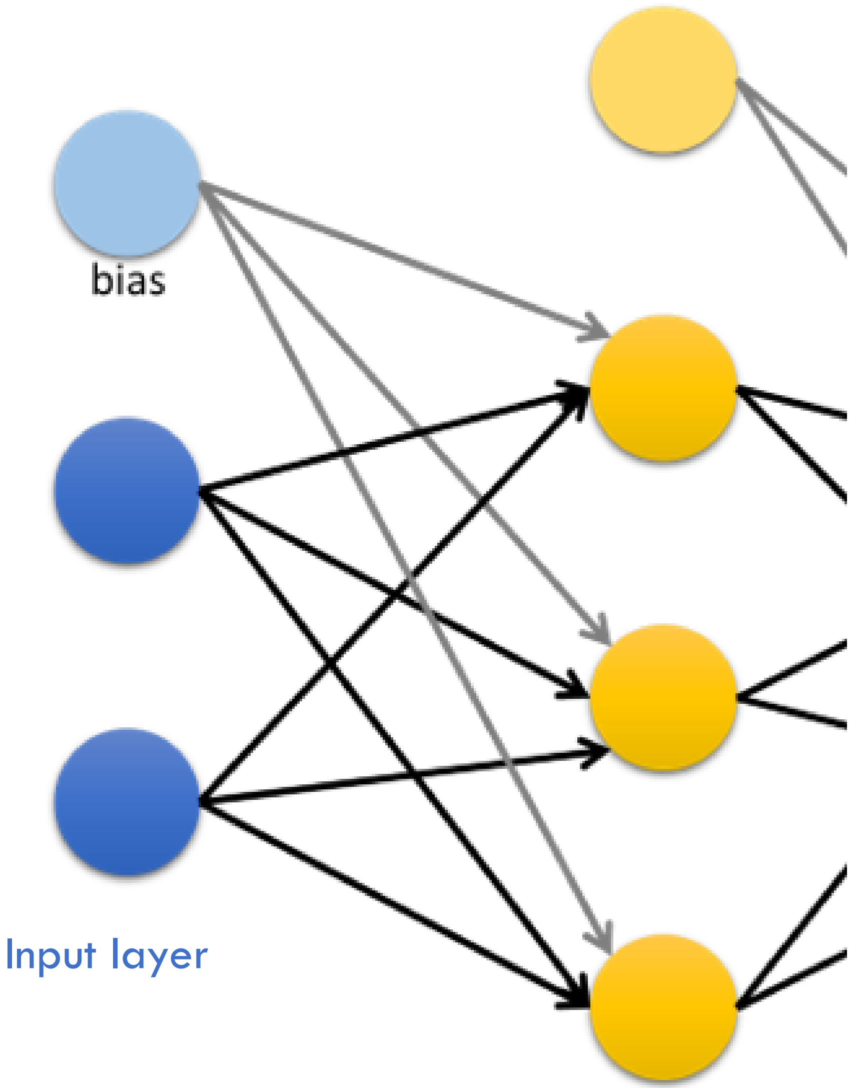
```
from tensorflow.keras import models
from tensorflow.keras.layers import Dense, Activation

##First definition

model = models.Sequential([
    Dense(20, input_shape=(10,)),
    Activation('sigmoid'),
    Dense(1)
])

model.summary()
```

The output layer is dense (fully connected), has one neuron and its activation function is linear.

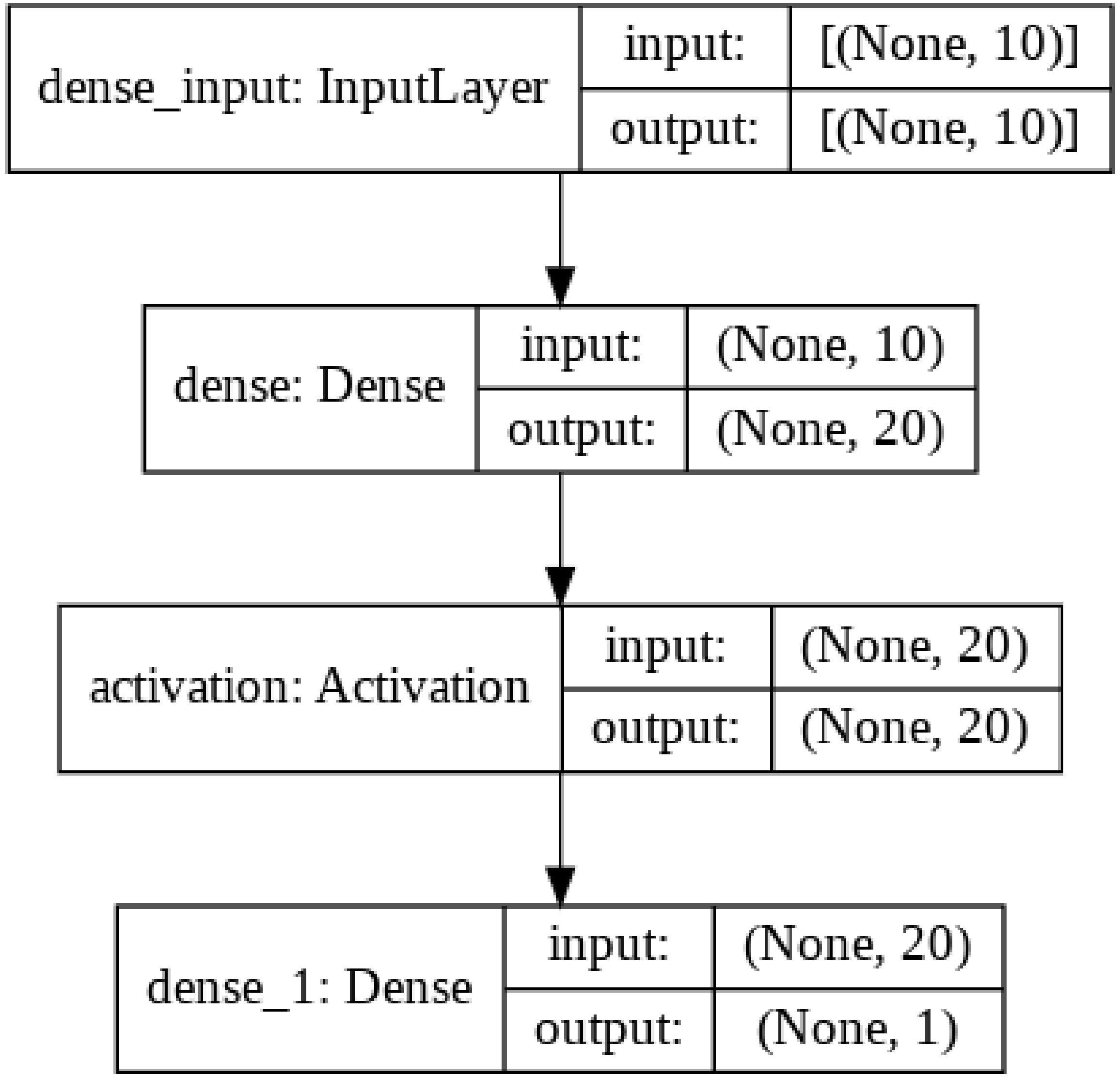
presents a summary of the main characteristics of the network

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                 (None, 20)               220
activation (Activation)       (None, 20)                0
dense_1 (Dense)               (None, 1)                 21
=================================================================
Total params: 241
Trainable params: 241
Non-trainable params: 0
_________________________________________________________________
```





$$\underbrace{20 \times 10}_{\text{Connection between neurons and input data}} + \underbrace{20}_{\text{Bias}} = 220$$




```
from keras.utils import plot_model
import pydot
plot_model(model, to_file = '/content/model.png', show_shapes = True)
```

SAME THING, DIFFERENT WAY…

```python
from keras import models
from keras import layers

##Second definition

model = models.Sequential()
model.add(layers.Dense(20, activation='sigmoid', input_shape=(10,)))
model.add(layers.Dense(1))

model.summary()
```

FUNCTION

```
def build_model(data_shape=(10,)):
    model = models.Sequential()
    model.add(layers.Dense(units=20, activation='sigmoid', input_shape=data_shape))
    model.add(layers.Dense(units=1))
    return model
```

```
model = build_model()
```

COMPILATION

The generation of the ANN is performed in the compilation stage, where the loss function, the training method and the metrics for the ANN evaluation are defined and configurated:

- The loss function mean_squared_error — How the network will be able to measure its performance on the training data, and thus how it will be able to steer itself in the right direction.
- The optimizer sgd — The mechanism through which the network will update itself based on the data it sees and its loss function.
- Metrics to monitor during training and testing mean_absolute_error, mean_absolute_percentage_error.

LOSS FUNCTION: MEAN SQUARED ERROR

$$E(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) = \sum_{j=1}^{n_y} \left(\hat{y}_j^{(i)} - y_j^{(i)} \right)^2 = \left\| \hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)} \right\|_2^2$$

$$J(\mathbf{W}, \mathbf{B}) = \frac{1}{m} \sum_{i=1}^{m} E(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n_y} \left(\hat{y}_j^{(i)} - y_j^{(i)} \right)^2 = \frac{1}{m} \sum_{i=1}^{m} \left\| \hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)} \right\|_2^2$$

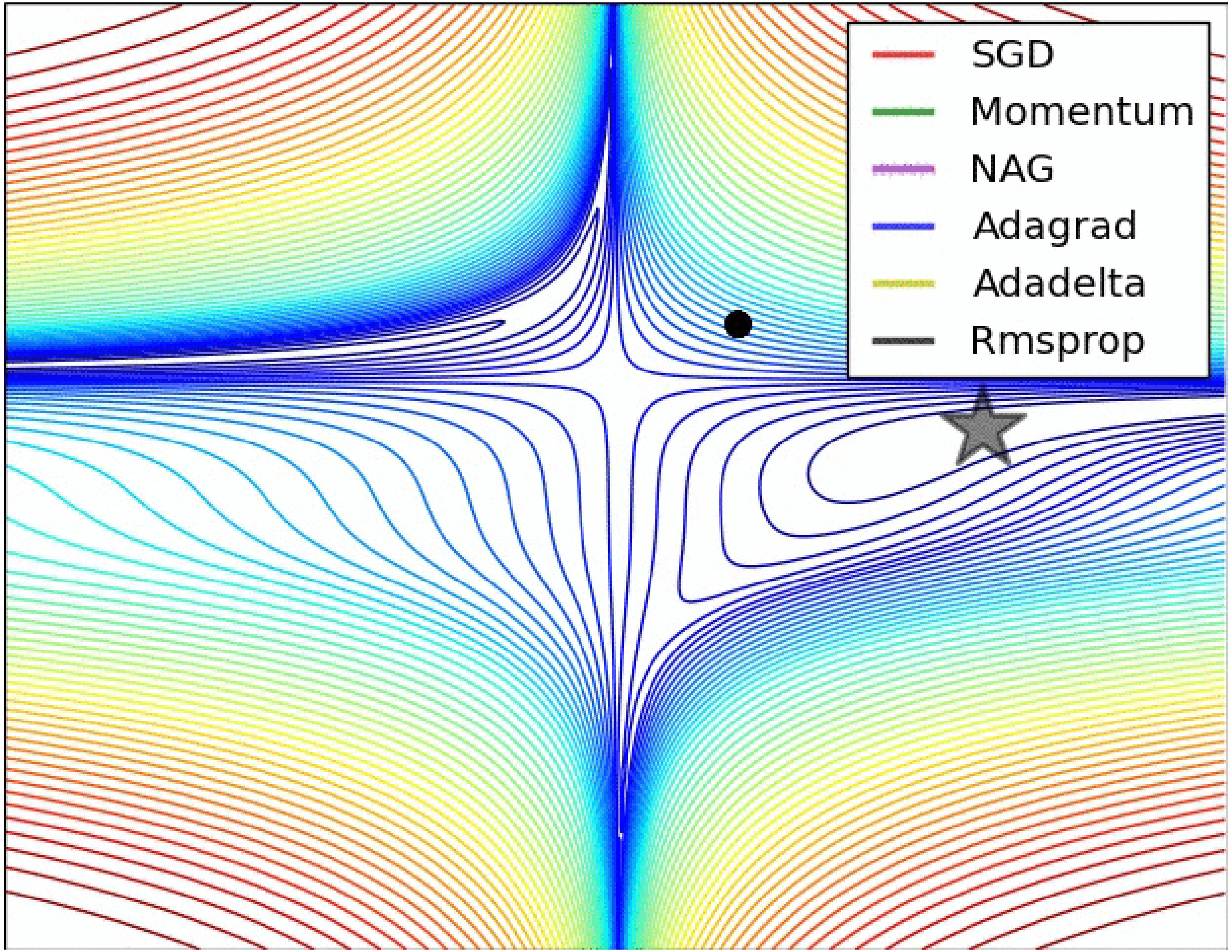
SGD

SGD is the same as gradient descent, except that it is used to split the data into batches. The parameter is called mini-batch size.

Faster optimizers are available in the literature to speed up the training step. We will apply the SGD + Momentum (known as SGD), but, be aware that are other popular Optimizer approaches such as Nesterov Accelerated Gradient, AdaGrad, RMSProp, Adam (ADAptive Moment estimation), and Nadam optimization.

The best optimizer, according to the literature, is Adam.

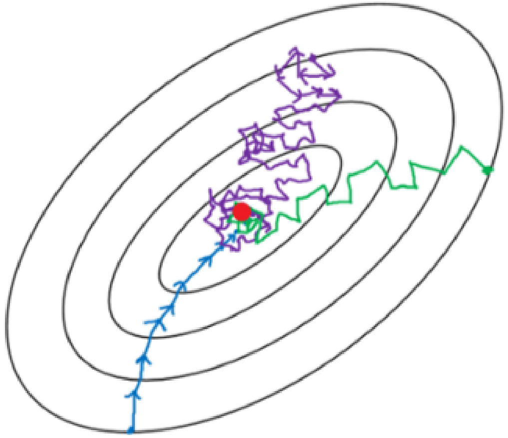
The SGD optimizer has a learning rate of 0.001 and momentum of 0.9.



Source: https://imgur.com/a/Hqolp#NKsFHJb

VARIAÇÕES DO GRADIENTE DESCENDENTE

Batch Gradient Descent, BGD: the gradient is calculated using **the entire training dataset** in each iteration, to update the parameters.

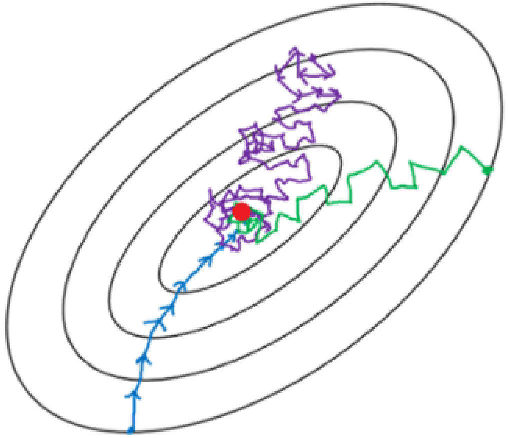


But if the number of training examples is large, then batch gradient descent is computationally very expensive! Imagine if you have 10000 data, each data with 10 features, there are 100 thousand values to compute at each iteration...

VARIAÇÕES DO GRADIENTE DESCENDENTE

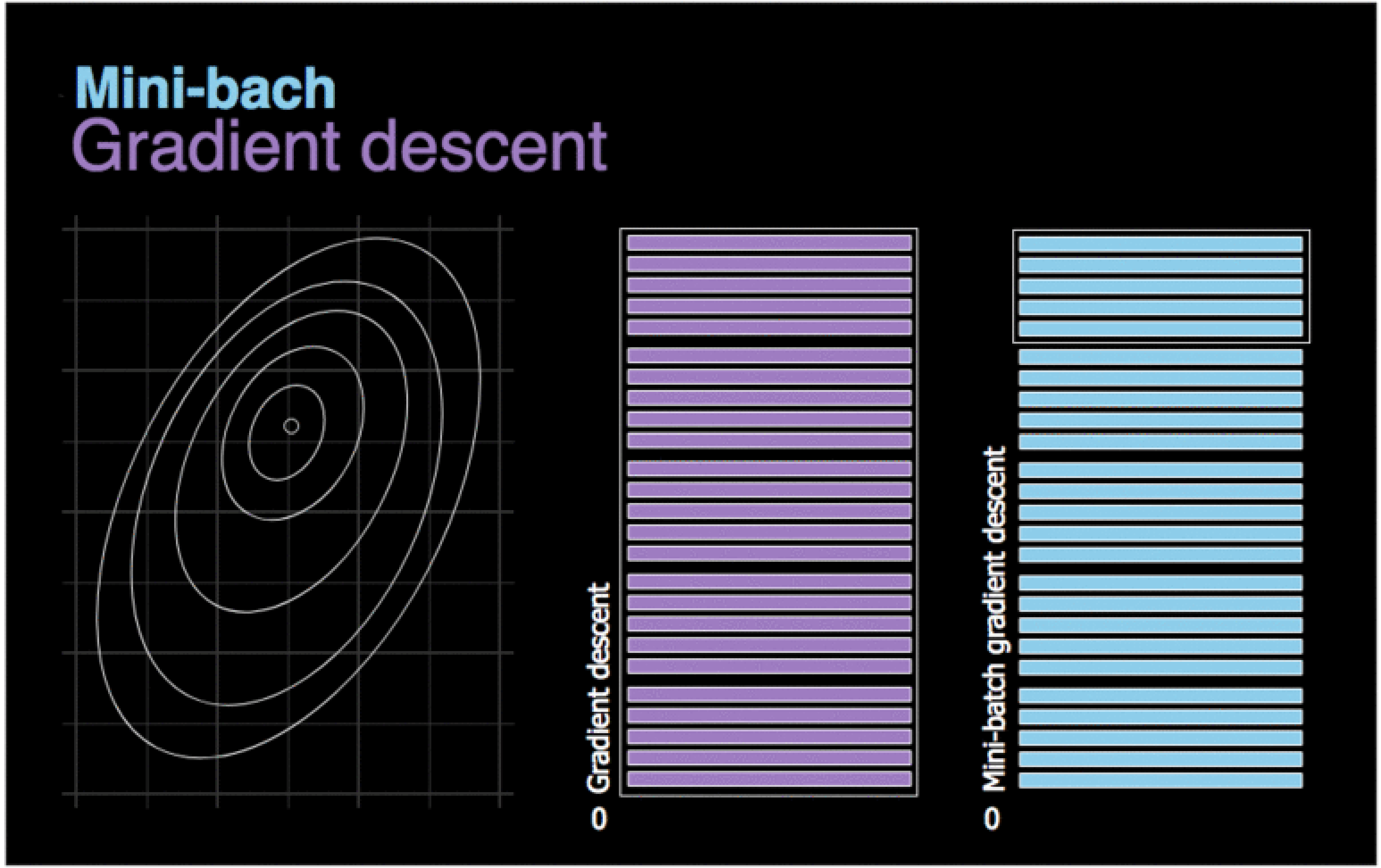
Batch Gradient Descent, BGD: the gradient is calculated using **the entire training dataset** in each iteration, to update the parameters.

But if the number of training examples is large, then batch gradient descent is computationally very expensive! Imagine if you have 10000 data, each data with 10 features, there are 100 thousand values to compute at each iteration...



Mini-batch Gradient Descent, MBGD: This is a type of gradient descent that works faster. The gradient is calculated using $b < m$ **data from the dataset** in each iteration, to update the parameters.

Stochastic Gradient Descent, SGD: the gradient is calculated using $b = 1$ **random training data** per iteration, to update the parameters. The SGD converges faster for larger data sets. However, as in SGD we only use one example at a time, we cannot use vectorized implementation. This can slow down the calculations.


Mini-bach
Gradient descent
Gradient descent
0
Mini-batch gradient descent
0

EPOCH

Batch Gradient Descent (BGD) We take the average of the gradients from all the training examples and use this average gradient to update our parameters.

Stochastic Gradient Descending (SGD) We take a training example for gradient calculation and use its gradient to update our parameters.

Mini Batch Gradient Descent (MBGD) The mini lot tries to find a balance between BGD and SGD.

For each epoch:

1. Use the training data: **BGD**, **SGD** ou **MBGD**
2. Calculate the gradient
3. Use the calculated gradient in to update the weights
4. Repeat steps 1 through 3 for all examples in the training dataset for the total number of epochs.

METRICS

Mean Absolute Error

$$MAE = \frac{1}{n}\sum_{1}^{n}|y^{(i)} - \hat{y}^{(i)}|$$

Mean squared error

$$MSE = \frac{1}{n}\sum_{i}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

Mean Absolute Percentage Error

$$MAPE = \frac{100}{n}\sum_{i}^{n}\frac{y^{(i)} - \hat{y}^{(i)}}{y^{(i)}}$$

FINALLY...

```
from keras import optimizers

sgd = optimizers.SGD(lr=0.001, momentum=0.9)

model.compile(optimizer=sgd,
              loss='mean_squared_error',
              metrics=['mean_absolute_error', 'mean_absolute_percentage_error'])
```

[99]

```
history_with_minibatch = model.fit(x_train_sca, y_train_sca, epochs=500, batch_size=32, verbose=2)

# To use the test loss history, comment the lines above and uncomment the lines below
#test_history_with_minibatch = TestLossHistory(x_val_sca, y_val_sca)
#history_with_minibatch = model.fit(x_train_sca, y_train_sca, epochs=10000, batch_size=32,
#                                    callbacks=[test_history_with_minibatch])
```

Saving the training process

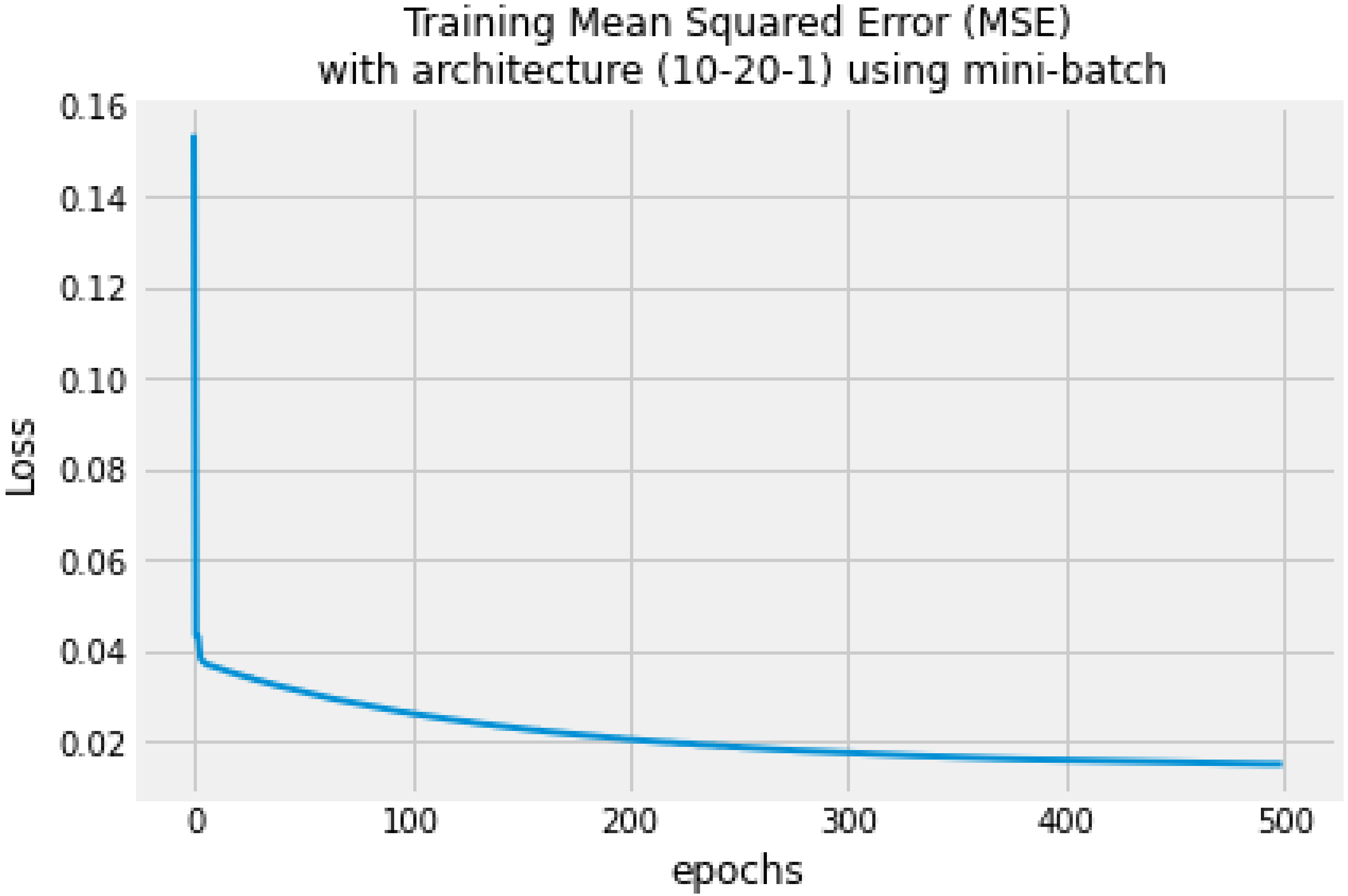
If the training process is saved, it is possible to graph the loss function, allowing a more detailed analysis of the process. For this we use:

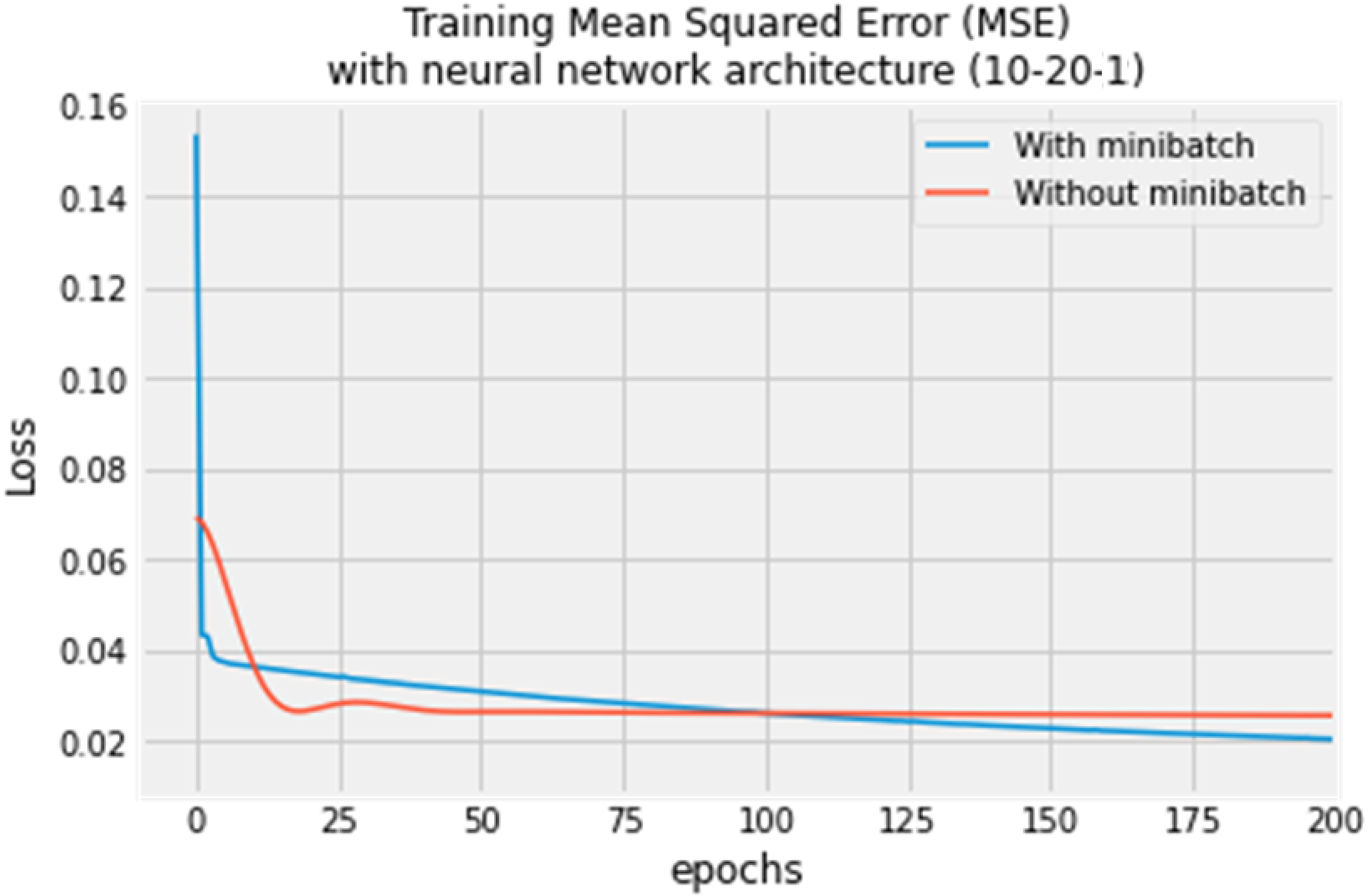
```
history_MODEL = model.fit (x_train, y_train, epochs = 1000)
```

In this training command the values of the cost function and the metric according to the seasons are saved in the `history_MODEL` object.

The `history_MODEL` object contains a dictionary with the values of the loss function and metrics for each epoch, which can be accessed using the following comment:

```
history_dict = history_MODEL.history
history_dict.keys ()
```


Training Mean Squared Error (MSE)
with architecture (10-20-1) using mini-batch
Loss
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0
100
200
300
400
500
epochs


Training Mean Squared Error (MSE)
with neural network architecture (10-20-1)
Loss
epochs
With minibatch
Without minibatch
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0
25
50
75
100
125
150
175
200

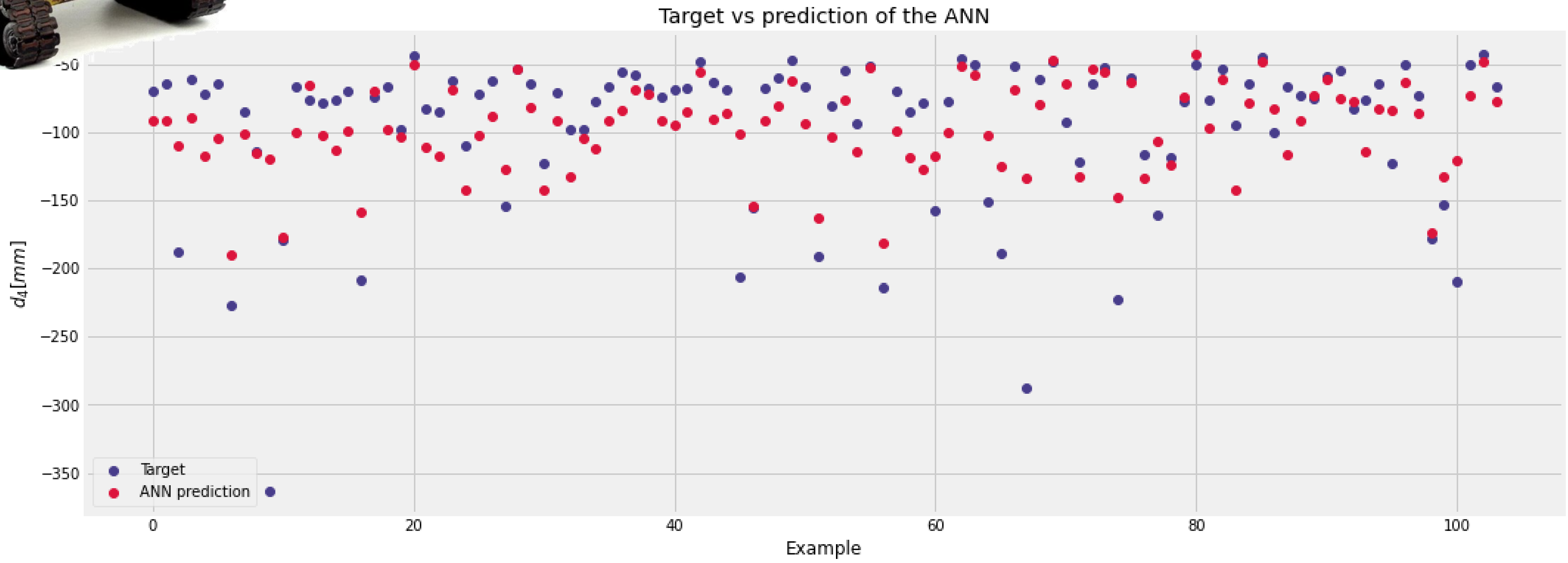
PERFORMANCE ANALYSIS

y_prev = model.predict(x_test)

```
pred_sca_train = model.predict(x_train_sca)
pred_sca_val = model.predict(x_val_sca)
print(pred_sca_val.shape,pred_sca_train.shape)
```

```
y_new_train = normalizer_y.inverse_transform(pred_sca_train)
y_new_val = normalizer_y.inverse_transform(pred_sca_val)
```

Hmmmmmm… the results are not so bad, for this simple network we made….



FOLLOW THE NOTEBOOK FOR THE NEXT STEPS

Changing number of neurons in the second intermediate;

Changing the activation function;

Changing optimizer.

Activation functions:
- sigmoid
- tanh
- softplus
- ReLU

Optimizers:
- SGD
- AdaGrad
- Adadelta
- RMSprop
- Adam

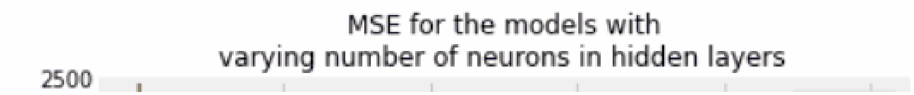
```python
# Creates a model with the specific number of neurons num_neurons and specific activation g
def make_model(num_neurons=20, g = 'sigmoid'):
  model = models.Sequential()
  model.add(layers.Dense(units=num_neurons, activation=g, input_shape=(10,)))
  model.add(layers.Dense(1))

  model.compile(optimizer=sgd,
                loss='mean_squared_error',
                metrics=['mean_absolute_error', 'mean_absolute_percentage_error'])
  return model
```

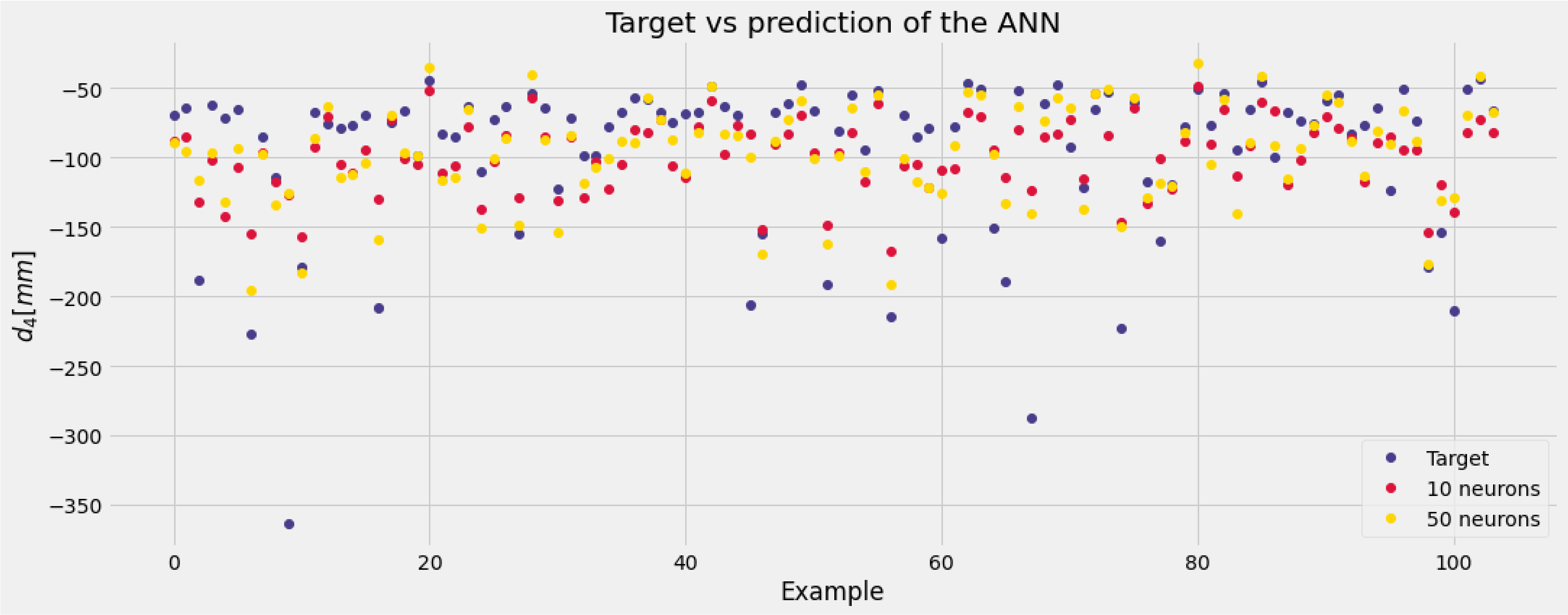
```python
model_10_neurons = make_model(num_neurons=10)
model_20_neurons = make_model(num_neurons=20)
model_30_neurons = make_model(num_neurons=30)
model_40_neurons = make_model(num_neurons=40)
model_50_neurons = make_model(num_neurons=50)
```

```
model_50_neurons = make_model(num_neurons=50)
```

```
# Training the models - output will be suppressed
print('10 neurons')
hist_10_neurons = model_10_neurons.fit(x_train_sca, y_train_sca, epochs=500, verbose=0)
print('20 neurons')
hist_20_neurons = model_20_neurons.fit(x_train_sca, y_train_sca, epochs=500, verbose=0)
print('30 neurons')
hist_30_neurons = model_30_neurons.fit(x_train_sca, y_train_sca, epochs=500, verbose=0)
print('40 neurons')
hist_40_neurons = model_40_neurons.fit(x_train_sca, y_train_sca, epochs=500, verbose=0)
print('50 neurons')
hist_50_neurons = model_50_neurons.fit(x_train_sca, y_train_sca, epochs=500, verbose=0)
print('Done!')
```

```
plt.title('MSE for the models with\nvarying number of neurons in hidden layers', fontsize=12)
plt.xlabel('epochs')
plt.ylabel('Loss')
plt.plot(hist_10_neurons.history['loss'], label='10',linewidth=1.0)
plt.plot(hist_20_neurons.history['loss'], label='20',linewidth=1.0)
plt.plot(hist_30_neurons.history['loss'], label='30',linewidth=1.0)
plt.plot(hist_40_neurons.history['loss'], label='40',linewidth=1.0)
plt.plot(hist_50_neurons.history['loss'], label='50',linewidth=1.0)
plt.ylim([0,2500])
plt.legend();
```

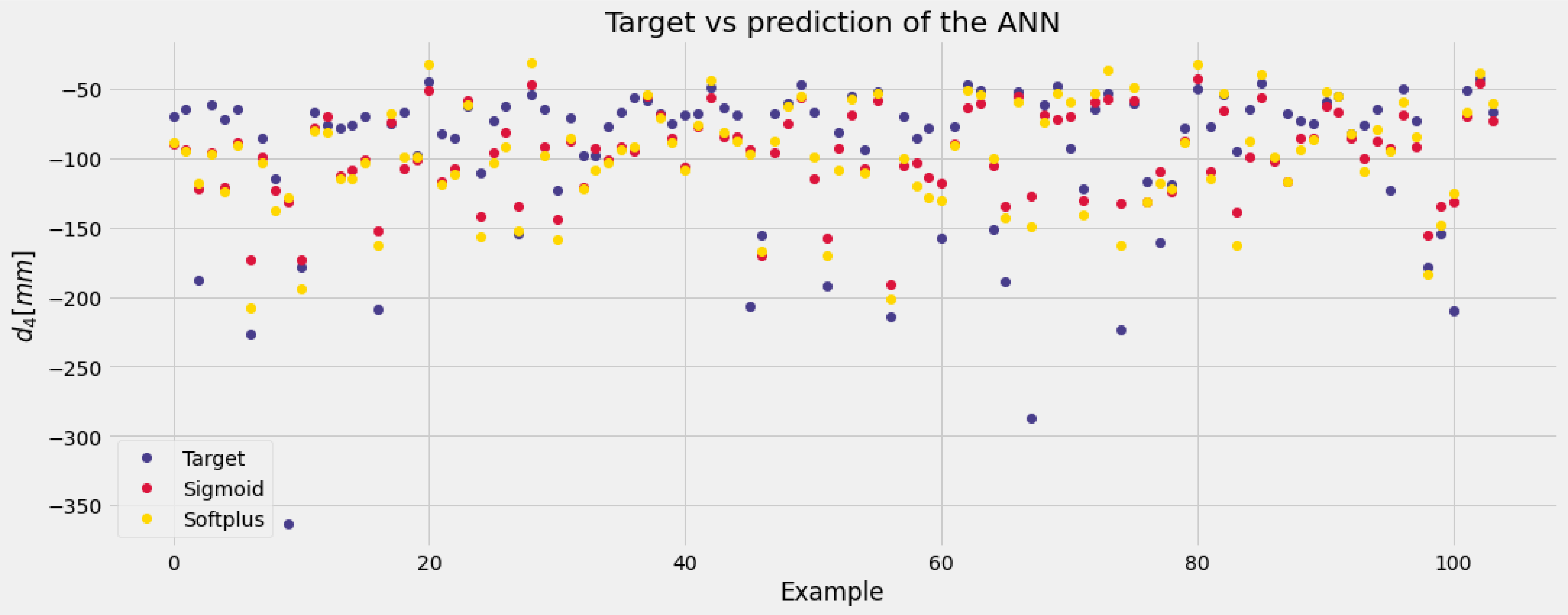



Target vs prediction of the ANN
$d_4[mm]$
Example
−50
−100
−150
−200
−250
−300
−350
0
20
40
60
80
100
Target
10 neurons
50 neurons

```python
sigmoid_model = make_model(g = 'sigmoid')
relu_model = make_model(g = 'relu')
tanh_model = make_model(g = 'tanh')
softplus_model = make_model(g = 'softplus')
```

```python
# Training the models
print('Sigmoid')
sigmoid_history = sigmoid_model.fit(x_train_sca, y_train_sca, epochs=500, batch_size=32, verbose = 0)
print('ReLU')
relu_history = relu_model.fit(x_train_sca, y_train_sca, epochs=500, batch_size=32, verbose = 0)
print('Tanh')
tanh_history = tanh_model.fit(x_train_sca, y_train_sca, epochs=500, batch_size=32, verbose = 0)
print('Softplus')
softplus_history = softplus_model.fit(x_train_sca, y_train_sca, epochs=500, batch_size=32, verbose = 0)
print('Done!!!')
```

```
Sigmoid
ReLU
Tanh
Softplus
Done!!!
```


Target vs prediction of the ANN
$d_4[mm]$
Example
Target
Sigmoid
Softplus
−50
−100
−150
−200
−250
−300
−350
0
20
40
60
80
100

```python
model_s = {'SGD': {}, 'AdaGrad': {}, 'Adadelta': {}, 'RMSprop': {}, 'Adam': {}}
activation_s =  ['sigmoid', 'tanh', 'softplus', 'relu']
```

```python
i = 0
for optimizer in model_s.keys():
    for activation in activation_s:
        print(f'Combination {i}: {optimizer} with {activation}')
        model = make_model_2(activation, optimizer)
        hist = model.fit(x_train_sca, y_train_sca, epochs=500, verbose=0)
        train_loss = hist.history['loss']
        val_loss = model.evaluate(x_val_sca, y_val_sca, verbose=0)
        model_s[optimizer][activation] = {'model': model, 'train': train_loss,
                                          'val': val_loss, 'hist': hist}
        i += 1
```

```
Combination 0: SGD with sigmoid
Combination 1: SGD with tanh
Combination 2: SGD with softplus
Combination 3: SGD with relu
Combination 4: AdaGrad with sigmoid
Combination 5: AdaGrad with tanh
Combination 6: AdaGrad with softplus
Combination 7: AdaGrad with relu
Combination 8: Adadelta with sigmoid
Combination 9: Adadelta with tanh
Combination 10: Adadelta with softplus
Combination 11: Adadelta with relu
Combination 12: RMSprop with sigmoid
Combination 13: RMSprop with tanh
Combination 14: RMSprop with softplus
Combination 15: RMSprop with relu
Combination 16: Adam with sigmoid
Combination 17: Adam with tanh
Combination 18: Adam with softplus
Combination 19: Adam with relu
```

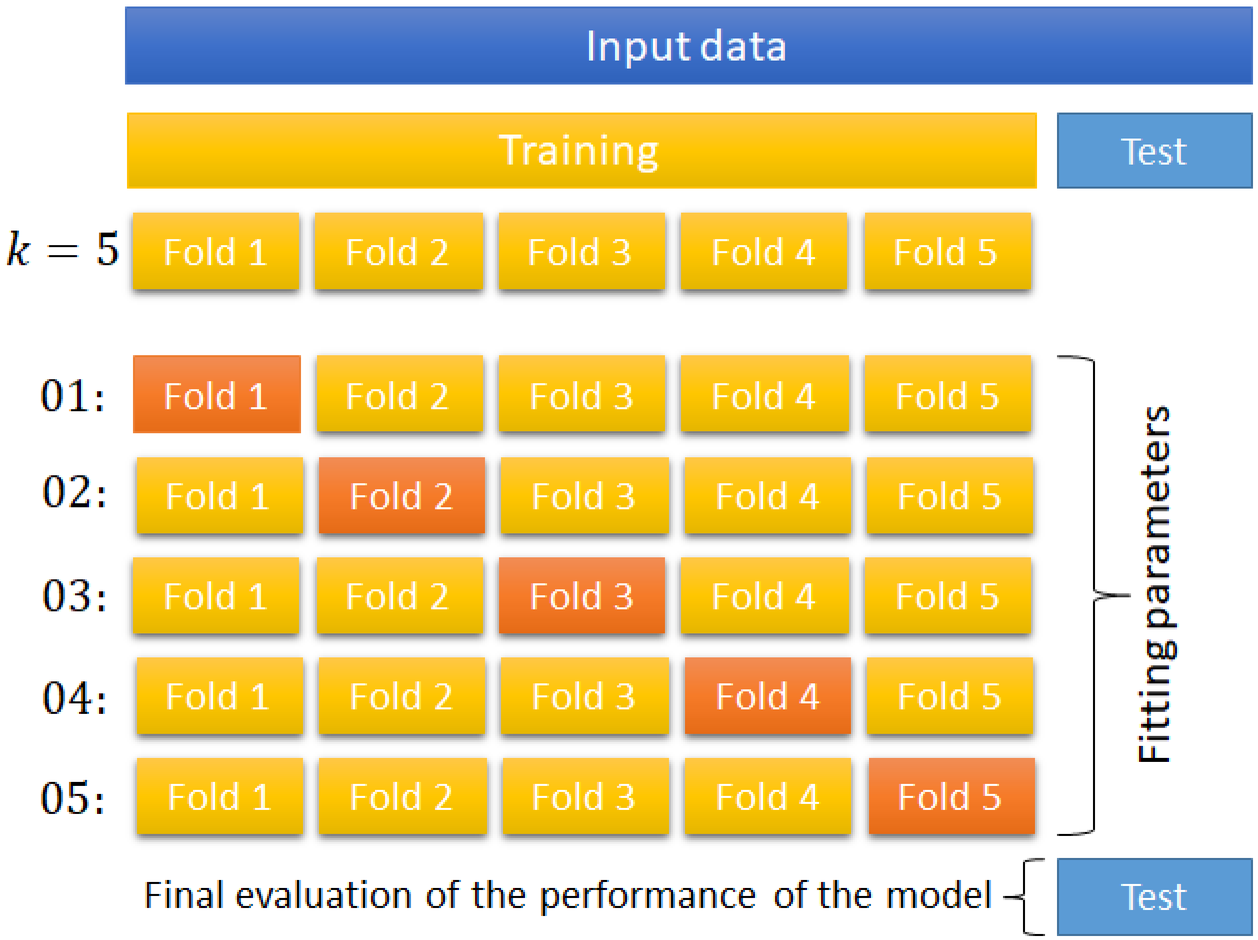
Your job

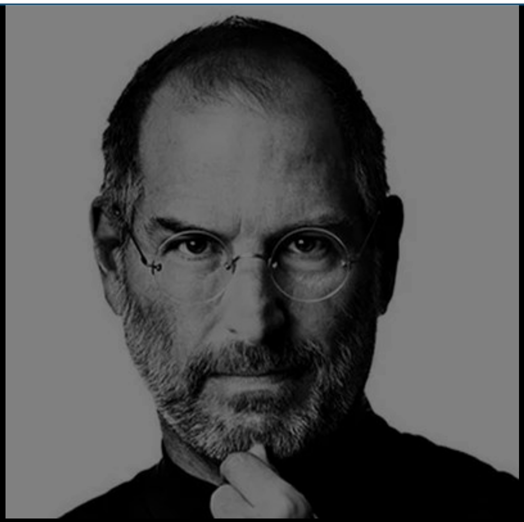
Review the Notebook.

Do the proposed homework.

Moodle, until 4/07, 23:59.

CROSS VALIDATION





THE END

"Have the courage to follow your heart and intuition. They somehow already know what you truly want to become. Everything else is secondary."
— **Steve Jobs**