In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [3]:

```
df1 = pd.read_csv("Data/ML_GRF_stance_N_subset.csv")
df1 = df1.drop('ID2',1)
df1
```

```
/var/folders/tg/lxlq3g6n3w5fk_7n_xq3hh380000gn/T/ipykernel_49411/23195
46634.py:2: FutureWarning: In a future version of pandas all arguments
of DataFrame.drop except for the argument 'labels' will be keyword-onl
y.
  df1 = df1.drop('ID2',1)
```

Out[3]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	3.591	2.30980	1.30420	1.544600	0.99642	-0.86461	-1.8383	-3.34520	-5.4749
1	2.199	0.30152	-0.49052	-0.002909	0.89121	0.30170	-2.7868	-6.95390	-9.065
2	4.317	1.07650	-0.75328	0.764090	-0.55040	-3.91910	-7.3576	-10.71500	-12.663
3	3.310	0.52531	1.30960	0.409730	-2.29800	-5.33850	-9.7976	-12.07200	-12.332
4	1.821	-1.24800	-5.39700	-6.074400	-5.49380	-8.26350	-9.4401	-9.57430	-9.199
...	...	...	...	...	...	...	...	...	..
12747	-0.434	4.50740	5.93060	-2.600900	-14.79000	-21.28800	-24.8200	-21.49100	-24.413
12748	1.164	4.43190	6.86710	-1.672000	-13.79200	-16.55900	-19.6420	-15.64400	-15.973
12749	4.382	6.59760	10.75200	9.721600	8.98050	1.87340	-6.8512	-11.07200	-13.695
12750	2.034	4.12090	9.56290	10.201000	4.41000	0.35724	-2.6013	-0.57981	-2.353
12751	1.835	4.16270	8.83130	6.386800	3.50040	-7.19020	-12.0380	-11.48800	-14.477

12752 rows × 100 columns

In [4]:

```
df2 = pd.read_csv("Data/ML_GRF_stance_N_outlier.csv")
df2 = df2.drop('ID2',1)
df2
```

```
/var/folders/tg/1xlq3g6n3w5fk_7n_xq3hh380000gn/T/ipykernel_49411/35857
51574.py:2: FutureWarning: In a future version of pandas all arguments
of DataFrame.drop except for the argument 'labels' will be keyword-onl
y.
  df2 = df2.drop('ID2',1)
```

Out[4]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	
0	0.021	3.0162	-0.81844	-8.89270	-10.48700	-12.3590	-14.4200	-18.7710	-19.3980	-
1	0.147	1.7636	-2.97610	-6.08910	-11.29600	-14.9640	-18.5100	-20.1410	-19.1230	-
2	1.593	-8.5376	-12.98600	-28.85900	-24.30000	-16.3350	-8.3953	-3.9667	-6.9380	
3	-3.543	-7.7014	-8.05010	-35.87200	-61.87800	-45.4830	-45.6890	-41.6360	-41.0160	-
4	0.368	-9.6953	-15.13000	-31.05200	-22.62400	-14.7320	-6.1482	-12.1620	-15.1730	-
...	...	...	...	...	...	...	...	...	...	
2939	0.200	2.9172	1.24840	-8.53980	-18.53100	-26.5580	-33.2530	-39.4330	-42.4730	-
2940	5.915	12.4600	23.89800	21.15500	0.48018	-12.6690	-18.0160	-23.8110	-27.7690	-
2941	-2.430	1.4202	4.81570	6.47910	5.62500	9.3313	11.8630	10.1870	3.7581	
2942	-1.106	-0.5330	-2.03000	-2.06990	-0.12273	-0.1550	-8.1966	-15.4030	-17.1340	-
2943	0.886	0.8837	0.24845	-0.78573	0.95121	-2.5983	-10.7050	-18.5510	-21.0860	-

2944 rows × 100 columns

In [5]:

```
from keras.layers import Dense,Conv2D,MaxPooling2D,UpSampling2D
from keras import Input, Model
from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from numpy.random import seed
seed(42)
from tensorflow import random
random.set_seed(42)
```

In [6]:

```
encoding_dim = 30
input_data = Input(shape=(100,))

# encoded representation of input
encoded = Dense(encoding_dim, activation='relu')(input_data)
encoded_2 = Dense(200, activation='relu')(encoded)
encoded_3 = Dense(300, activation='relu')(encoded_2)
encoded_4 = Dense(encoding_dim, activation='relu')(encoded_3)
decoded_2 = Dense(200, activation='relu')(encoded_4)
decoded_1 = Dense(300, activation='relu')(decoded_2)
x2 = Dense(300, activation='relu')(decoded_1)
# decoded representation of code
decoded = Dense(100)(x2)
# Model which take input image and shows decoded images
autoencoder = Model(input_data, decoded)
```

```
Metal device set to: Apple M1

2022-05-05 16:58:49.470677: I tensorflow/core/common_runtime/pluggable
_device/pluggable_device_factory.cc:305] Could not identify NUMA node
of platform GPU ID 0, defaulting to 0. Your kernel may not have been b
uilt with NUMA support.
2022-05-05 16:58:49.472115: I tensorflow/core/common_runtime/pluggable
_device/pluggable_device_factory.cc:271] Created TensorFlow device (/j
ob:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physic
al PluggableDevice (device: 0, name: METAL, pci bus id: <undefined>)
```

In [7]:

```
autoencoder.compile(optimizer='adam', loss='mse')
```

In [20]:

```
data1 = np.array(df1)
data2 = np.array(df2)
```

In [21]:

```
X_train = data1
X_test = data2
```

In [22]:

```
print(X_train.shape)
print(X_test.shape)
```

```
(12752, 100)
(2944, 100)
```

In [23]:

```
n_epochs = 800
```

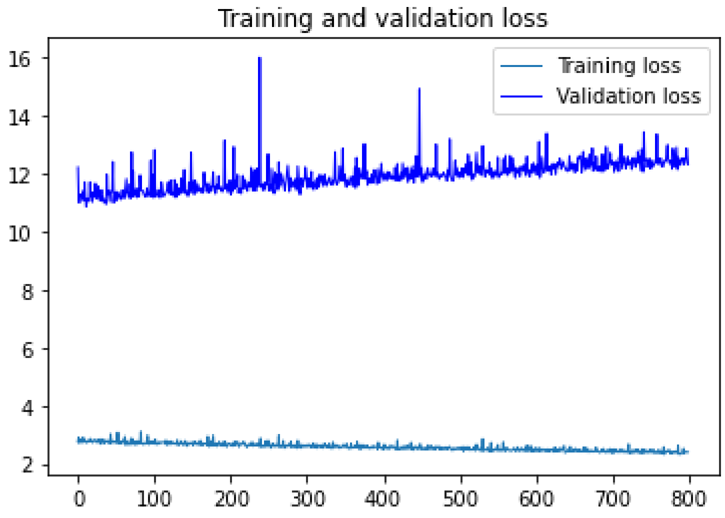
In [24]:

```
history = autoencoder.fit(X_train, X_train,
                epochs=n_epochs,
                batch_size=256,
                validation_data=(X_test, X_test),
                         verbose = False)
```

```
2022-05-05 21:52:10.142443: I tensorflow/core/grappler/optimizers/cust
om_graph_optimizer_registry.cc:113] Plugin optimizer for device_type G
PU is enabled.
2022-05-05 21:52:12.543746: I tensorflow/core/grappler/optimizers/cust
om_graph_optimizer_registry.cc:113] Plugin optimizer for device_type G
PU is enabled.
```

In [25]:

```
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(n_epochs)
plt.figure()
plt.plot(epochs, loss, '-', label='Training loss', lw=1)
plt.plot(epochs, val_loss, 'b', label='Validation loss', lw=1)
plt.title('Training and validation loss')
plt.legend()
plt.show()
plt.close()
```



In [26]:

```
decoded_data = autoencoder(X_test)
```

In [27]:

```
decoded_data.shape
```

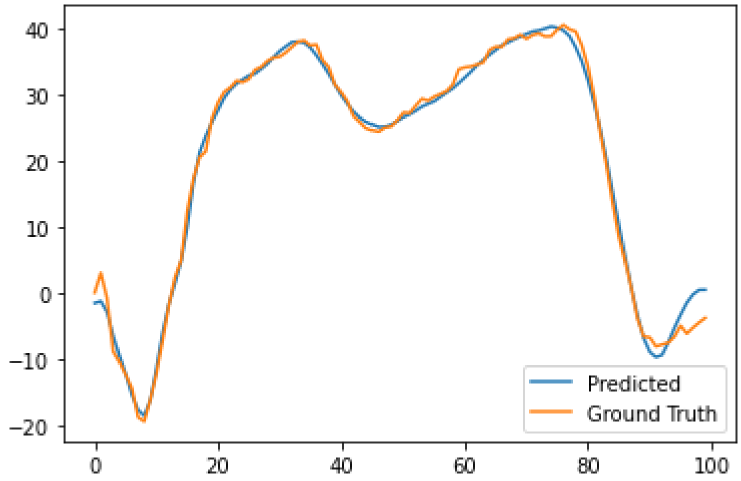
Out[27]:

```
TensorShape([2944, 100])
```

Example 1

In [28]:

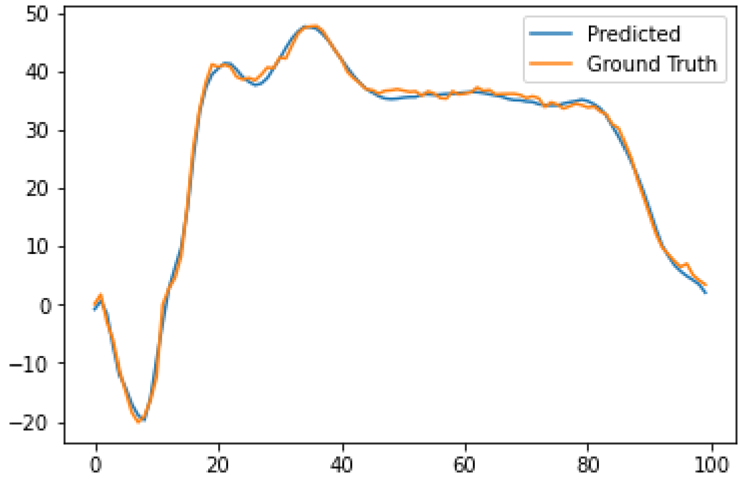
```
xx = np.arange(0,100)
plt.plot(xx, decoded_data[0], label="Predicted")
plt.plot(xx, X_test[0], label="Ground Truth")
plt.legend()
plt.show()
```



Example 2

In [29]:

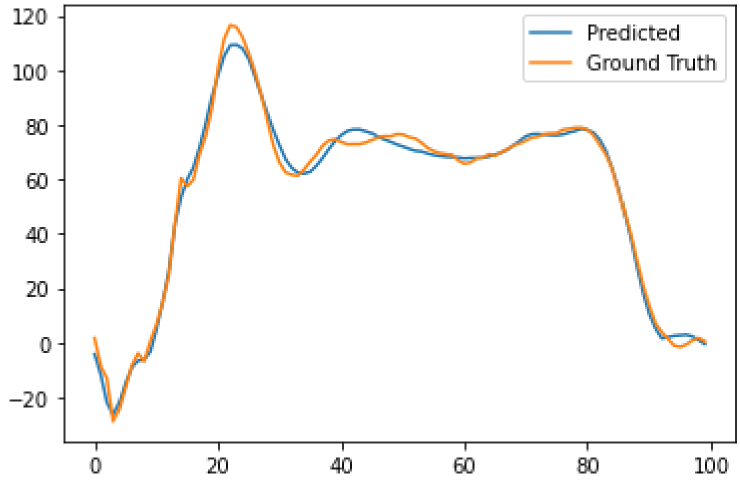
```
xx = np.arange(0,100)
plt.plot(xx, decoded_data[1], label="Predicted")
plt.plot(xx, X_test[1], label="Ground Truth")
plt.legend()
plt.show()
```



Example 3

In [30]:

```
xx = np.arange(0,100)
plt.plot(xx, decoded_data[2], label="Predicted")
plt.plot(xx, X_test[2], label="Ground Truth")
plt.legend()
plt.show()
```



In [31]:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from math import sqrt

#mean_squared_error(X_test, decoded_data)

r2 = r2_score(X_test, decoded_data)
rmse = sqrt(mean_squared_error(X_test, decoded_data))

# RMSE normalised by mean:
nrmse = rmse/sqrt(np.mean(X_test**2))
```

In [32]:

```
r2
```

Out[32]:

```
0.9410645223730196
```

In [33]:

```
rmse
```

Out[33]:

```
3.5106538368030464
```

In [34]:

```
nrmse
```

Out[34]:

```
0.07488062981513163
```

In [ ]:

```

```