In [33]:

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

In [54]:

```
df = pd.read_csv("Data/ML_GRF_stance_N.csv")
df
```

Out[54]:

	3.591	2.3098	1.3042	1.5446	0.99642	-0.86461	-1.8383	-3.3452	-5.474(
0	2.199	0.30152	-0.49052	-0.002909	0.89121	0.30170	-2.7868	-6.95390	-9.065(
1	4.317	1.07650	-0.75328	0.764090	-0.55040	-3.91910	-7.3576	-10.71500	-12.663(
2	3.310	0.52531	1.30960	0.409730	-2.29800	-5.33850	-9.7976	-12.07200	-12.332(
3	1.821	-1.24800	-5.39700	-6.074400	-5.49380	-8.26350	-9.4401	-9.57430	-9.199(
4	2.466	1.43950	-1.35490	-4.770000	-4.33850	-4.46170	-4.4079	-4.11200	-4.262(
...	...	...	...	...	...	...	...	...	..
15690	-0.434	4.50740	5.93060	-2.600900	-14.79000	-21.28800	-24.8200	-21.49100	-24.413(
15691	1.164	4.43190	6.86710	-1.672000	-13.79200	-16.55900	-19.6420	-15.64400	-15.973(
15692	4.382	6.59760	10.75200	9.721600	8.98050	1.87340	-6.8512	-11.07200	-13.695(
15693	2.034	4.12090	9.56290	10.201000	4.41000	0.35724	-2.6013	-0.57981	-2.353(
15694	1.835	4.16270	8.83130	6.386800	3.50040	-7.19020	-12.0380	-11.48800	-14.477(

15695 rows × 100 columns

In [40]:

```
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 [70]:

```
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)
```

In [71]:

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

In [72]:

```
data = np.array(df)
data.shape
```

Out[72]:

```
(15695, 100)
```

In [73]:

```
from sklearn.model_selection import train_test_split

X_train, X_test = train_test_split(data, test_size=0.2, random_state=2022)
```

In [74]:

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

```
(12556, 100)
(3139, 100)
```

In [75]:

```
n_epochs = 800
```

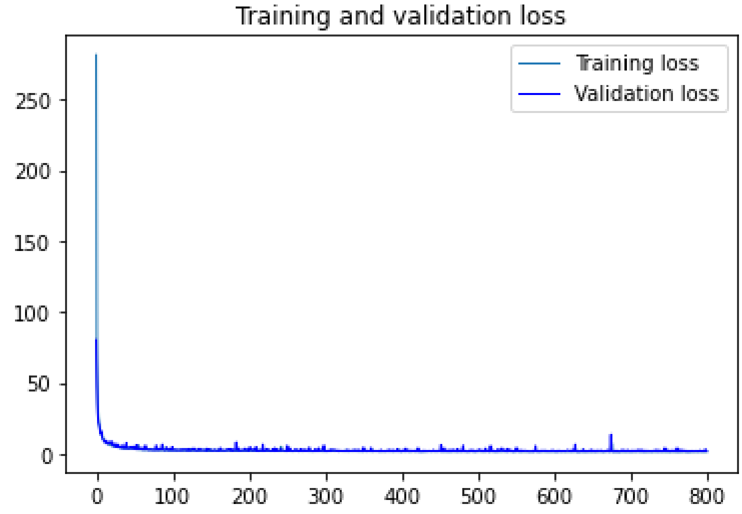
In [76]:

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

```
2022-05-05 22:22:34.068142: I tensorflow/core/grappler/optimizers/cust
om_graph_optimizer_registry.cc:113] Plugin optimizer for device_type G
PU is enabled.
2022-05-05 22:22:34.686956: I tensorflow/core/grappler/optimizers/cust
om_graph_optimizer_registry.cc:113] Plugin optimizer for device_type G
PU is enabled.
```

In [77]:

```
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 [78]:

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

In [79]:

```
decoded_data.shape
```

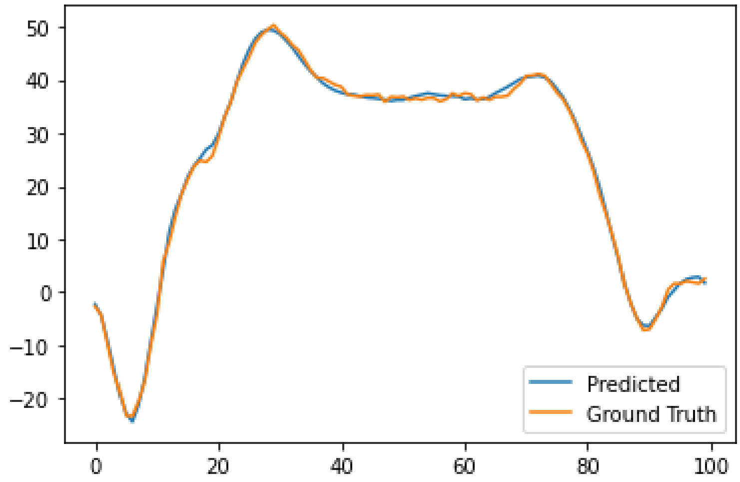
Out[79]:

```
TensorShape([3139, 100])
```

Example 1:

In [80]:

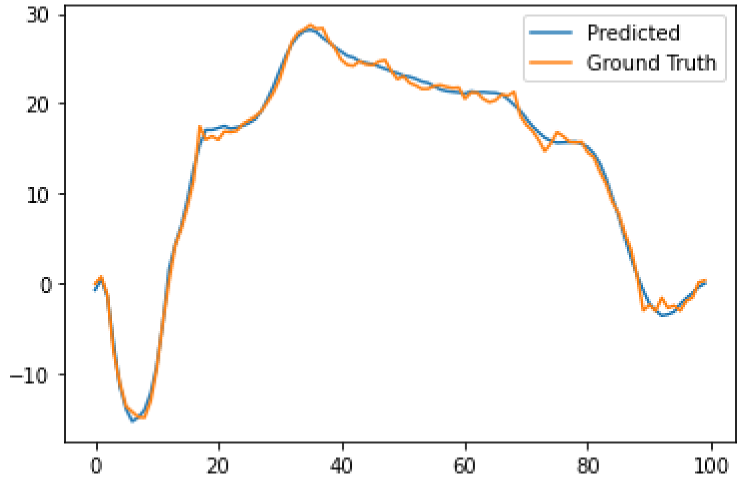
```
xx = np.arange(0,100)
plt.plot(xx, decoded_data[0], label="Predicted")
plt.plot(xx, X_test[0], label="Ground Truth")
plt.legend()
plt.show()
```



Example 2:

In [81]:

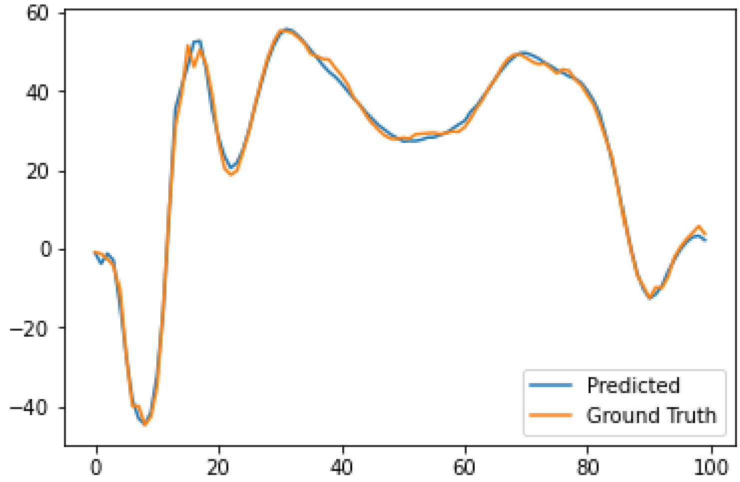
```
xx = np.arange(0,100)
plt.plot(xx, decoded_data[1], label="Predicted")
plt.plot(xx, X_test[1], label="Ground Truth")
plt.legend()
plt.show()
```



Example 3:

In [82]:

```
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 [83]:

```
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))


#reference: https://stats.stackexchange.com/questions/194278/meaning-of-reconstruct
```

Out[83]:

```
0.039040845854724705
```

Assume X_test is your original data and decoded_data is the compressed data What I usually use as the measure of reconstruction error (in the context of PCA, but also other methods) is the coefficient of determination $R2$ and the Root Mean Squared Error (or normalised RMSE). These two are easy to compute and give you a quick idea of what the reconstruction did.

Since $R2$ = 1.0 for a perfect fit, you can judge the reconstruction by how close the $R2$ is to 1.0. In our case, our R2 = 0.96

In [84]:

```
r2
```

Out[84]:

```
0.9737635397550299
```

In [85]:

```
rmse
```

Out[85]:

```
1.3718608947024753
```

In [86]:

```
nrmse
```

Out[86]:

```
0.039040845854724705
```

In [ ]:

```

```