parallel_rss_model_all_data_fine-Copy1

September 16, 2024

[1]:
```
import h5py
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.python.framework.ops import EagerTensor
from tensorflow.python.ops.resource_variable_ops import ResourceVariable
import time
import pandas as pd
import scipy.io
import math
import sklearn
import sklearn.datasets
import tensorflow.keras as keras
from keras.models import Sequential
from keras.layers import Dense
from keras_visualizer import visualizer

dfx = pd.read_csv(r'D:\Ashish Google Drive\Seventh␣
 ↪Sem\PhD\4th_paper_ml\python\Final_dataset_fine\data_output_all_combined_fine.
 ↪txt')
dfy = pd.read_csv(r'D:\Ashish Google Drive\Seventh␣
 ↪Sem\PhD\4th_paper_ml\python\Final_dataset_fine\data_input_all_combined_fine.
 ↪txt')

#print(dfx)
#print(dfy)
```

[2]:
```
from sklearn.model_selection import train_test_split
X_train,X_test = train_test_split(dfx,test_size=0.3,random_state=0)
```

[3]:
```
from sklearn.model_selection import train_test_split
Y_train,Y_test = train_test_split(dfy,test_size=0.3,random_state=0)
```

[4]:
```
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras import layers

#train
X_train.head()
```

```
X_train_norm_features = X_train.copy()
X_train_norm_features = np.array(X_train_norm_features)

Y_train_norm_labels = Y_train.copy()
Y_train_norm_labels = np.array(Y_train_norm_labels)


#test
X_test_norm_features = X_test.copy()
X_test_norm_features = np.array(X_test_norm_features)

Y_test_norm_labels = Y_test.copy()
Y_test_norm_labels = np.array(Y_test_norm_labels)
```

```
X_train_norm.copy()           -39.982  -47.84  -33.913  -9.368  -2.968  22.978
10866346   -4.508  -9.588   11.907   0.741  12.987  10.707
8350807    18.353  12.409  -23.351 -27.215  23.637  10.057
11962099   -2.786  16.327  -12.920   6.053  31.695  35.432
7449500     7.200  19.850  -24.389  -4.811  15.891  20.969
6489633   -12.492  -6.499  -34.726 -27.319 -11.180 -12.544
…            …     …       …       …       …       …
10638075  -31.586 -30.872   16.176   8.634   0.142   0.626
5157699     8.381  18.882  -30.882  -8.957   1.107  17.660
2215104   -14.904 -28.782   -2.804  -4.793  15.434  24.051
1484405     7.594  -9.104   -5.600  -4.577  15.655  20.647
8325804    10.388  13.524  -35.457 -30.695  10.906   6.617

[8583239 rows x 6 columns]
X_train_norm_features [[ -4.508  -9.588  11.907   0.741  12.987  10.707]
 [ 18.353  12.409 -23.351 -27.215  23.637  10.057]
 [ -2.786  16.327 -12.92    6.053  31.695  35.432]
 …
 [-14.904 -28.782  -2.804  -4.793  15.434  24.051]
 [  7.594  -9.104  -5.6    -4.577  15.655  20.647]
 [ 10.388  13.524 -35.457 -30.695  10.906   6.617]]
Y_train_norm_labels [[  4 -14   7  -4  -2   4]
 [ 16  18   6   6 -10   6]
 [ 19  10  19  -8 -10 -10]
 …
 [-10 -20   0  -8 -10   0]
 [-14  -6   8  -2 -10   2]
 [ 16  16  -2   6 -10   0]]
Y_test_norm.copy()          -20  -20.1  -20.2  -10  -10.1  -10.2
9460651   -17    -20    -11    2      0     -6
10521359   -2     -2    -20   -4      2    -10
3652691    -4    -10     18    2      8     -6
8083701    16     -8     18    4    -10     -6
7591956    14    -12      0    8      0      4
```

```
…       …    …    …  …    …    …
5988974    6     4   -14    8     4     0
6289334    8   -10     0   -4    -4     0
5079849    2     0    16    6     0   -10
9916060  -11    -8    13  -10     2    -8
8897826   20   -14     8    6    -6     4

[3678532 rows x 6 columns]
X_test_norm_features          -39.982  -47.84  -33.913  -9.368  -2.968  22.978
9460651   -10.292 -10.560   -5.794  -1.618 -23.774   3.227
10521359  -37.162 -17.353  -18.699   0.022 -22.304  -5.684
3652691     5.551  16.327   22.268  23.904  -7.799   5.850
8083701    13.681  24.510  -11.623 -10.161  17.583  31.059
7591956     8.830  10.023    0.288 -20.925  -2.338  -2.972
…             …      …       …       …      …       …
5988974     0.626   6.008  -12.031 -18.507 -27.711 -29.157
6289334   -12.311 -10.214   -2.112  -6.380  10.024  11.437
5079849    12.372  27.465    0.925  11.914  -4.663  12.482
9916060   -16.776  -9.643   12.225  27.693   9.456  21.676
8897826    10.246  11.354   -1.845 -22.265  15.171  15.288

[3678532 rows x 6 columns]
```

[5]:
```
import keras_tuner
```

```
Using TensorFlow backend
```

[20]:
```
def build_regressor(hp):
    model = keras.Sequential(
        [
            layers.Dense(units=hp.Int("units", 6, 256, 32), activation="tanh"),
            layers.Dense(units=6),
        ]
    )
    model.compile(
        optimizer="adam",
        loss="mean_squared_error",
        # Objective is one of the metrics.
        metrics=[keras.metrics.MeanAbsoluteError()],
    )
    return model


tuner = keras_tuner.RandomSearch(
    hypermodel=build_regressor,
    objective=keras_tuner.Objective("val_mean_absolute_error", direction="min"),
    max_trials=6,
    overwrite=True,
```

```
    directory="./",
    project_name="built_in_metrics_all_fine_2",
)

tuner.search(X_train_norm_features, Y_train_norm_labels,validation_split = 0.
 ↪15, epochs=10)

tuner.results_summary()
```

```
Trial 6 Complete [00h 44m 52s]
val_mean_absolute_error: 0.07545071840286255

Best val_mean_absolute_error So Far: 0.057804595679044724
Total elapsed time: 04h 21m 38s
Results summary
Results in ./built_in_metrics_all_fine_1
Showing 10 best trials
Objective(name="val_mean_absolute_error", direction="min")

Trial 4 summary
Hyperparameters:
units: 230
Score: 0.057804595679044724

Trial 1 summary
Hyperparameters:
units: 166
Score: 0.06618934124708176

Trial 3 summary
Hyperparameters:
units: 134
Score: 0.07078669220209122

Trial 5 summary
Hyperparameters:
units: 102
Score: 0.07545071840286255

Trial 0 summary
Hyperparameters:
units: 70
Score: 0.07765627652406693

Trial 2 summary
Hyperparameters:
units: 6
Score: 0.551080584526062
```

[21]:
```
tuner.search_space_summary()
```

```
Search space summary
Default search space size: 1
units (Int)
{'default': None, 'conditions': [], 'min_value': 6, 'max_value': 256, 'step':
32, 'sampling': 'linear'}
```

[22]:
```
# Get the top 2 models.
models = tuner.get_best_models(num_models=2)
best_model = models[0]
# Build the model.
# Needed for `Sequential` without specified `input_shape`.
best_model.build(input_shape=(None, 6))
best_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 230)               1610

 dense_1 (Dense)             (None, 6)                 1386

=================================================================
Total params: 2996 (11.70 KB)
Trainable params: 2996 (11.70 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[23]:
```
tuner.results_summary()
```

```
Results summary
Results in ./built_in_metrics_all_fine_1
Showing 10 best trials
Objective(name="val_mean_absolute_error", direction="min")

Trial 4 summary
Hyperparameters:
units: 230
Score: 0.057804595679044724

Trial 1 summary
Hyperparameters:
units: 166
Score: 0.06618934124708176

Trial 3 summary
Hyperparameters:
```

```
units: 134
Score: 0.07078669220209122

Trial 5 summary
Hyperparameters:
units: 102
Score: 0.07545071840286255

Trial 0 summary
Hyperparameters:
units: 70
Score: 0.07765627652406693

Trial 2 summary
Hyperparameters:
units: 6
Score: 0.551080584526062
```

[24]:
```
best_model.fit(X_train_norm_features, Y_train_norm_labels,validation_split = 0.
↪15, epochs=10)
```

```
Epoch 1/10
227993/227993 [==============================] - 251s 1ms/step - loss: 0.0089 -
mean_absolute_error: 0.0679 - val_loss: 0.0130 - val_mean_absolute_error: 0.0800
Epoch 2/10
227993/227993 [==============================] - 243s 1ms/step - loss: 0.0089 -
mean_absolute_error: 0.0677 - val_loss: 0.0122 - val_mean_absolute_error: 0.0760
Epoch 3/10
227993/227993 [==============================] - 247s 1ms/step - loss: 0.0088 -
mean_absolute_error: 0.0675 - val_loss: 0.0116 - val_mean_absolute_error: 0.0788
Epoch 4/10
227993/227993 [==============================] - 245s 1ms/step - loss: 0.0088 -
mean_absolute_error: 0.0675 - val_loss: 0.0070 - val_mean_absolute_error: 0.0604
Epoch 5/10
227993/227993 [==============================] - 246s 1ms/step - loss: 0.0088 -
mean_absolute_error: 0.0674 - val_loss: 0.0096 - val_mean_absolute_error: 0.0702
Epoch 6/10
227993/227993 [==============================] - 249s 1ms/step - loss: 0.0088 -
mean_absolute_error: 0.0676 - val_loss: 0.0092 - val_mean_absolute_error: 0.0688
Epoch 7/10
227993/227993 [==============================] - 247s 1ms/step - loss: 0.0088 -
mean_absolute_error: 0.0676 - val_loss: 0.0108 - val_mean_absolute_error: 0.0757
Epoch 8/10
227993/227993 [==============================] - 249s 1ms/step - loss: 0.0088 -
mean_absolute_error: 0.0676 - val_loss: 0.0088 - val_mean_absolute_error: 0.0658
Epoch 9/10
227993/227993 [==============================] - 249s 1ms/step - loss: 0.0087 -
mean_absolute_error: 0.0674 - val_loss: 0.0100 - val_mean_absolute_error: 0.0741
```

```
Epoch 10/10
227993/227993 [==============================] - 249s 1ms/step - loss: 0.0087 -
mean_absolute_error: 0.0673 - val_loss: 0.0090 - val_mean_absolute_error: 0.0692
```

[24]:
```
<keras.src.callbacks.History at 0x194d1b44190>
```

[13]:
```
#plt.plot(best_model['loss'], label='train')
#plt.plot(best_model['val_loss'], label='test')
#plt.legend()
#plt.title('Training History'),
#plt.xlabel('Epoch'),
#plt.ylabel('Validation Loss')
#plt.show()
```

[25]:
```
best_model.save('all_model_2.keras')
best_model.save('all_model_2.h5')

model_json = best_model.to_json()
with open("all_model_2.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
best_model.save_weights("all_model_2.h5")
print("Saved model to disk")
```

```
Saved model to disk

C:\Users\Hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\keras\src\engine\training.py:3079: UserWarning: You are saving your
model as an HDF5 file via `model.save()`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

[26]:
```
predict12 = best_model.predict(X_train_norm_features)
#predict_train_X = norm_model8.predict(X_train_norm_features)
#print("predict_train_X",predict_train_X)
#denorm_predict_train_X = np.multiply(y_norm_train,predict_train_X)
#print("denorm_predict_train_X",denorm_predict_train_X)
print(predict12)
print(np.shape(predict12))
print(Y_train_norm_labels)
print(type(Y_train_norm_labels))
print(np.shape(Y_train_norm_labels))
#Y_train_norm_labels = Y_train_norm_labels.reshape(137,1)
error12 = predict12-Y_train_norm_labels
print(np.shape(Y_train_norm_labels))
print(error12)
#print(np.shape(error8))
print(np.mean(np.square(np.abs(error12))))
```

```
#plt(history)
```

```
268227/268227 [==============================] - 199s 740us/step
[[ 3.98273325e+00 -1.41698942e+01  7.02920389e+00 -3.96829557e+00
  -2.04198885e+00  3.97841716e+00]
 [ 1.58094006e+01  1.75955372e+01  6.18313789e+00  5.94216681e+00
  -9.92908955e+00  5.88854694e+00]
 [ 1.90149002e+01  9.83222294e+00  1.89996948e+01 -8.09838676e+00
  -9.99605465e+00 -9.98972225e+00]
 …
 [-9.97579575e+00 -1.99920197e+01  1.61714107e-02 -7.87847996e+00
  -1.00445576e+01  8.23003054e-02]
 [-1.39123783e+01 -6.04278421e+00  8.06994724e+00 -1.94951534e+00
  -1.00191841e+01  1.92616343e+00]
 [ 1.58520432e+01  1.57699795e+01 -1.72332728e+00  5.98136711e+00
  -9.94271183e+00  4.75108624e-04]]
(8583239, 6)
[[  4 -14   7  -4  -2   4]
 [ 16  18   6   6 -10   6]
 [ 19  10  19  -8 -10 -10]
 …
 [-10 -20   0  -8 -10   0]
 [-14  -6   8  -2 -10   2]
 [ 16  16  -2   6 -10   0]]
<class 'numpy.ndarray'>
(8583239, 6)
(8583239, 6)
[[-1.72667503e-02 -1.69894218e-01  2.92038918e-02  3.17044258e-02
  -4.19888496e-02 -2.15828419e-02]
 [-1.90599442e-01 -4.04462814e-01  1.83137894e-01 -5.78331947e-02
   7.09104538e-02 -1.11453056e-01]
 [ 1.49002075e-02 -1.67777061e-01 -3.05175781e-04 -9.83867645e-02
   3.94535065e-03  1.02777481e-02]
 …
 [ 2.42042542e-02  7.98034668e-03  1.61714107e-02  1.21520042e-01
  -4.45575714e-02  8.23003054e-02]
 [ 8.76216888e-02 -4.27842140e-02  6.99472427e-02  5.04846573e-02
  -1.91841125e-02 -7.38365650e-02]
 [-1.47956848e-01 -2.30020523e-01  2.76672721e-01 -1.86328888e-02
   5.72881699e-02  4.75108624e-04]]
0.009009736494002307
```

[ ]:
```
predict12a = best_model.predict(X_test_norm_features)
#predict_train_X = norm_model8.predict(X_train_norm_features)
#print("predict_train_X",predict_train_X)
#denorm_predict_train_X = np.multiply(y_norm_train,predict_train_X)
#print("denorm_predict_train_X",denorm_predict_train_X)
```

```
print(predict12a)

print(np.shape(predict12a))
print(Y_test_norm_labels)
print(type(Y_test_norm_labels))
print(np.shape(Y_test_norm_labels))
#Y_train_norm_labels = Y_train_norm_labels.reshape(137,1)
error12a = predict12a-Y_test_norm_labels
print(np.shape(Y_test_norm_labels))
print(error12a)
print(np.shape(error12a))
print(np.mean(np.square(np.abs(error12a))))
#plt(history)
```

[ ]:
```
model_json = best_model.to_json()
with open("all_model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
best_model.save_weights("all_model.h5")
print("Saved model to disk")
```

[ ]:
```
json_file = open('all_model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
```

[ ]:
```
best_model.save('all_model.keras')
```

[13]:
```
new_model = tf.keras.models.load_model('all_model_1.keras')
```

[14]:
```
new_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 230)               1610

 dense_1 (Dense)             (None, 6)                 1386

=================================================================
Total params: 2996 (11.70 KB)
Trainable params: 2996 (11.70 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:
```
best_model.save('all_model.h5')
```

[4]:
```
new_model1 = tf.keras.models.load_model('all_model_1.keras', compile=False)
```

[5]:
```
new_model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 230)               1610

 dense_1 (Dense)             (None, 6)                 1386

=================================================================
Total params: 2996 (11.70 KB)
Trainable params: 2996 (11.70 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[31]:
```
dfx_check = pd.read_csv(r'D:\Ashish Google Drive\Seventh␣
 ↪Sem\PhD\4th_paper_ml\python\DataSets\data_output_3_2.txt')
dfy_check = pd.read_csv(r'D:\Ashish Google Drive\Seventh␣
 ↪Sem\PhD\4th_paper_ml\python\DataSets\data_input_3_2.txt')
```

[40]:
```
from sklearn.model_selection import train_test_split
X_train,X_test = train_test_split(dfx_check,test_size=0.3,random_state=0)

from sklearn.model_selection import train_test_split
Y_train,Y_test = train_test_split(dfy_check,test_size=0.3,random_state=0)
```

[41]:
```
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras import layers
X_train.head()
X_train_norm_features = X_train.copy()
print("X_train_norm.copy()",X_train.copy())
Y_train_norm_labels = Y_train.copy()
Y_train_norm_labels = np.array(Y_train_norm_labels)
X_train_norm_features = np.array(X_train_norm_features)
print("X_train_norm_features",X_train_norm_features)
print("Y_train_norm_labels",Y_train_norm_labels)
X_test_norm_features = X_test.copy()
Y_test_norm_labels = Y_test.copy()
print("Y_test_norm.copy()",Y_test.copy())
#Y_test_norm_labels = Y_test_norm_labels.to_numpy()
X_test_norm_features = np.array(X_test_norm_features)
Y_test_norm_labels = np.array(Y_test_norm_labels)
#X_test_norm_features = X_test.copy()
print("X_test_norm_features",X_test)
```

```
X_train_norm.copy()       0  0.1  0.2  0.3  0.4  0.5
6    7    0    0    0    0    0
```

```
13  14    0    0    0    0    0
4    5    0    0    0    0    0
2    3    0    0    0    0    0
5    6    0    0    0    0    0
14  15    0    0    0    0    0
9   10    0    0    0    0    0
7    8    0    0    0    0    0
16  17    0    0    0    0    0
11  12    0    0    0    0    0
3    4    0    0    0    0    0
0    1    0    0    0    0    0
15  16    0    0    0    0    0
12  13    0    0    0    0    0
X_train_norm_features [[ 7  0  0  0  0  0]
 [14  0  0  0  0  0]
 [ 5  0  0  0  0  0]
 [ 3  0  0  0  0  0]
 [ 6  0  0  0  0  0]
 [15  0  0  0  0  0]
 [10  0  0  0  0  0]
 [ 8  0  0  0  0  0]
 [17  0  0  0  0  0]
 [12  0  0  0  0  0]
 [ 4  0  0  0  0  0]
 [ 1  0  0  0  0  0]
 [16  0  0  0  0  0]
 [13  0  0  0  0  0]]
Y_train_norm_labels [[  7   7  19   2   4   4]
 [ -5   1  -5  -4   8   6]
 [  1   1  10   8  -6   6]
 …
 [ 13  -8 -11  -8 -10   0]
 [  1   4 -20  -2 -10   2]
 [-17   1  13   4  -8  -6]]
Y_test_norm.copy()          -20  -20.1  -20.2  -10  -10.1  -10.2
2558706   19    -20     10    4    -10      4
2208349   13    -11     10   -4      0    -10
1426390    1    -11     16   -4      8     -8
1621354    4    -11     13   -4      0      0
1406908    1    -14     -2    8    -10      8
…       …     …     …  …     …     …
2015101   10    -11     19   -8    -10     -6
953357    -8     16    -17   -4      0      6
1519020    1     10      1  -10     -6     -8
633929   -11    -11    -11    8     -4    -10
1313148   -2      7     13   -8     -2      8

[823200 rows x 6 columns]
```

```
X_test_norm_features      0  0.1  0.2  0.3  0.4  0.5
18  19    0    0    0    0    0
1    2    0    0    0    0    0
19  20    0    0    0    0    0
8    9    0    0    0    0    0
10  11    0    0    0    0    0
17  18    0    0    0    0    0
```

[34]:
```
predict12 = new_model1.predict(X_train_norm_features)
#predict_train_X = norm_model8.predict(X_train_norm_features)
#print("predict_train_X",predict_train_X)
#denorm_predict_train_X = np.multiply(y_norm_train,predict_train_X)
#print("denorm_predict_train_X",denorm_predict_train_X)
print(predict12)
print(np.shape(predict12))
print(Y_train_norm_labels)
print(type(Y_train_norm_labels))
print(np.shape(Y_train_norm_labels))
#Y_train_norm_labels = Y_train_norm_labels.reshape(137,1)
error12 = predict12-Y_train_norm_labels
print(np.shape(Y_train_norm_labels))
print(error12)
#print(np.shape(error8))
print(np.mean(np.square(np.abs(error12))))
#plt(history)
```

```
60025/60025 [==============================] - 49s 821us/step
[[  7.045394     6.967816    19.077456     1.9879396    4.0535655
    3.9502766 ]
 [ -5.2469463    1.0126202   -5.0328045   -3.9583404    7.935346
    6.0668917 ]
 [  0.87774295   0.9591308   10.026771     7.989295    -6.043178
    5.896037  ]
 …
 [ 12.998318    -7.9460073  -10.965882    -7.8988934  -10.039447
    0.06163019]
 [  0.8453657    3.9354253  -20.042774    -1.8691561  -10.124183
    1.9683435 ]
 [-16.859388     0.88292783  12.974099     4.003473    -8.045867
   -6.059412  ]]
(1920799, 6)
[[  7   7  19   2   4   4]
 [ -5   1  -5  -4   8   6]
 [  1   1  10   8  -6   6]
 …
 [ 13  -8 -11  -8 -10   0]
 [  1   4 -20  -2 -10   2]
 [-17   1  13   4  -8  -6]]
```

```
<class 'numpy.ndarray'>
(1920799, 6)
(1920799, 6)
[[ 0.04539394 -0.03218412  0.07745552 -0.0120604   0.0535655  -0.04972339]
 [-0.24694633  0.01262021 -0.03280449  0.04165959 -0.06465387  0.06689167]
 [-0.12225705 -0.04086918  0.02677059 -0.01070499 -0.04317808 -0.1039629 ]
 …
 [-0.00168228  0.05399275  0.0341177   0.10110664 -0.03944683  0.06163019]
 [-0.1546343  -0.06457472 -0.0427742   0.13084388 -0.1241827  -0.0316565 ]
 [ 0.14061165 -0.11707217 -0.02590084  0.00347281 -0.04586697 -0.059412  ]]
0.008920830023362894
```

[35]:
```
predict12a = new_model1.predict(X_test_norm_features)
#predict_train_X = norm_model8.predict(X_train_norm_features)
#print("predict_train_X",predict_train_X)
#denorm_predict_train_X = np.multiply(y_norm_train,predict_train_X)
#print("denorm_predict_train_X", denorm_predict_train_X)
print(predict12a)

print(np.shape(predict12a))
print(Y_test_norm_labels)
print(type(Y_test_norm_labels))
print(np.shape(Y_test_norm_labels))
#Y_train_norm_labels = Y_train_norm_labels.reshape(137,1)
error12a = predict12a-Y_test_norm_labels
print(np.shape(Y_test_norm_labels))
print(error12a)
print(np.shape(error12a))
print(np.mean(np.square(np.abs(error12a))))
#plt(history)
```

```
25725/25725 [==============================] - 19s 752us/step
[[ 18.819689   -19.907509    10.01861      4.0113683  -10.062325
    3.937744  ]
 [ 13.080879   -11.196361    10.00459     -4.0012503   -0.09037608
   -9.99972   ]
 [  1.1062601  -11.170524    16.086462    -4.0443482    7.942209
   -7.8918486 ]
 …
 [  1.0602305    9.702753     1.1198127   -9.98101     -6.000906
   -8.060306  ]
 [-11.080045   -10.91409    -11.012386     8.105999    -3.8805215
  -10.114193  ]
 [ -2.1043534    6.87991     13.081214    -8.032471    -2.1050594
    7.9799547 ]]
(823200, 6)
[[ 19 -20  10   4 -10   4]
 [ 13 -11  10  -4   0 -10]
```

```
 [  1 -11  16  -4   8  -8]
 …
 [  1  10   1 -10  -6  -8]
 [-11 -11 -11   8  -4 -10]
 [ -2   7  13  -8  -2   8]]
<class 'numpy.ndarray'>
(823200, 6)
(823200, 6)
[[-1.80311203e-01  9.24911499e-02  1.86100006e-02  1.13682747e-02
  -6.23245239e-02 -6.22560978e-02]
 [ 8.08792114e-02 -1.96360588e-01  4.59003448e-03 -1.25026703e-03
  -9.03760791e-02  2.80380249e-04]
 [ 1.06260061e-01 -1.70523643e-01  8.64620209e-02 -4.43482399e-02
  -5.77912331e-02  1.08151436e-01]
 …
 [ 6.02304935e-02 -2.97246933e-01  1.19812727e-01  1.89895630e-02
  -9.05990601e-04 -6.03055954e-02]
 [-8.00447464e-02  8.59098434e-02 -1.23863220e-02  1.05998993e-01
   1.19478464e-01 -1.14192963e-01]
 [-1.04353428e-01 -1.20090008e-01  8.12139511e-02 -3.24707031e-02
  -1.05059385e-01 -2.00452805e-02]]
(823200, 6)
0.008934312423057348
```

[23]:
```
plt.plot(new_model1.history['loss'])
plt.plot(new_model1.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[23], line 1
----> 1 plt.plot(new_model1.history['loss'])
      2 plt.plot(new_model1.history['val_loss'])
      3 plt.title('model loss')

TypeError: 'History' object is not subscriptable
```

[6]:
```
j_val= pd.read_csv(r'D:\Ashish Google Drive\Seventh␣
 ↪Sem\PhD\4th_paper_ml\python\validate\joint1.txt')
```

[8]:
```
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras import layers
```

```
j_val.head()
j_val_norm_features = j_val.copy()
#print("j_val_norm.copy()",j_val.copy())
j_val_norm_features = np.array(j_val_norm_features)
print("j_val_norm_features",j_val_norm_features)
print("j_val_shape",np.shape(j_val))
```

```
j_val_norm_features [[ 1  0  0  0  0  0]
 [ 2  0  0  0  0  0]
 [ 3  0  0  0  0  0]
 [ 4  0  0  0  0  0]
 [ 5  0  0  0  0  0]
 [ 6  0  0  0  0  0]
 [ 7  0  0  0  0  0]
 [ 8  0  0  0  0  0]
 [ 9  0  0  0  0  0]
 [10  0  0  0  0  0]
 [11  0  0  0  0  0]
 [12  0  0  0  0  0]
 [13  0  0  0  0  0]
 [14  0  0  0  0  0]
 [15  0  0  0  0  0]
 [16  0  0  0  0  0]
 [17  0  0  0  0  0]
 [18  0  0  0  0  0]
 [19  0  0  0  0  0]
 [20  0  0  0  0  0]]
j_val_shape (20, 6)
```

[9]:
```
predict12a = new_model1.predict(j_val_norm_features)

print(predict12a)

print(np.shape(predict12a))

#plt(history)
```

```
1/1 [==============================] - 0s 170ms/step
[[ -1.1244261    0.3311052    2.3444054    0.13440591  -0.15608305
    0.22917116]
 [ -2.1868036    0.5127536    2.5655348    0.29968756  -0.30188364
    0.4412458 ]
 [ -3.2855194    0.6929818    2.7781472    0.46150094  -0.45086998
    0.65019524]
 [ -4.364477     0.87604487   2.9793143    0.61728126  -0.5937211
    0.86381125]
 [ -5.4284225    1.05996      3.1717136    0.77157027  -0.7301032
    1.0855012 ]
```

```
 [ -6.522745     1.2406223    3.3603013    0.93041736  -0.86720294
    1.3115559 ]
 [ -7.630296     1.4210577    3.5460904    1.0925899   -1.0056775
    1.5364754 ]
 [ -8.713226     1.6070437    3.7269533    1.2536659   -1.1422923
    1.7578521 ]
 [ -9.816599     1.797044     3.9023683    1.4152267   -1.2814488
    1.9771683 ]
 [-10.975772     1.9885356    4.0732584    1.5798028   -1.424406
    2.2005541 ]
 [-12.114705     2.1864862    4.239405     1.7431915   -1.5591667
    2.4352508 ]
 [-13.194851     2.3922246    4.400227     1.9034963   -1.6822653
    2.678908  ]
 [-14.302092     2.597243     4.5557265    2.06641     -1.8087406
    2.9195635 ]
 [-15.467853     2.7984676    4.7052474    2.23222     -1.9442332
    3.1522837 ]
 [-16.607662     3.0032256    4.8464       2.392235    -2.0747974
    3.386411  ]
 [-17.724266     3.2114024    4.9749618    2.545282    -2.1966677
    3.6309931 ]
 [-18.900156     3.4145553    5.0896344    2.6989145   -2.3201647
    3.8837597 ]
 [-20.0862       3.6130273    5.196701     2.8535326   -2.4427636
    4.1367493 ]
 [-21.23248      3.8084521    5.298637     3.0073874   -2.561035
    4.383404  ]
 [-22.44708      3.9939146    5.3856797    3.1640573   -2.6859055
    4.6256094 ]]
(20, 6)
```

[10]:
```
df = pd.DataFrame(predict12a)

print(df)
```

```
            0         1         2         3         4         5
0   -1.124426  0.331105  2.344405  0.134406 -0.156083  0.229171
1   -2.186804  0.512754  2.565535  0.299688 -0.301884  0.441246
2   -3.285519  0.692982  2.778147  0.461501 -0.450870  0.650195
3   -4.364477  0.876045  2.979314  0.617281 -0.593721  0.863811
4   -5.428422  1.059960  3.171714  0.771570 -0.730103  1.085501
5   -6.522745  1.240622  3.360301  0.930417 -0.867203  1.311556
6   -7.630296  1.421058  3.546090  1.092590 -1.005677  1.536475
7   -8.713226  1.607044  3.726953  1.253666 -1.142292  1.757852
8   -9.816599  1.797044  3.902368  1.415227 -1.281449  1.977168
9  -10.975772  1.988536  4.073258  1.579803 -1.424406  2.200554
10 -12.114705  2.186486  4.239405  1.743191 -1.559167  2.435251
```

```
11 -13.194851  2.392225  4.400227  1.903496 -1.682265  2.678908
12 -14.302092  2.597243  4.555727  2.066410 -1.808741  2.919564
13 -15.467853  2.798468  4.705247  2.232220 -1.944233  3.152284
14 -16.607662  3.003226  4.846400  2.392235 -2.074797  3.386411
15 -17.724266  3.211402  4.974962  2.545282 -2.196668  3.630993
16 -18.900156  3.414555  5.089634  2.698915 -2.320165  3.883760
17 -20.086201  3.613027  5.196701  2.853533 -2.442764  4.136749
18 -21.232479  3.808452  5.298637  3.007387 -2.561035  4.383404
19 -22.447081  3.993915  5.385680  3.164057 -2.685905  4.625609
```

[11]:
```
df.to_csv('val.txt', index=False)
```

[12]:
```
x =df.to_numpy()
print(x)
```

```
[[ -1.1244261    0.3311052    2.3444054    0.13440591  -0.15608305
    0.22917116]
 [ -2.1868036    0.5127536    2.5655348    0.29968756  -0.30188364
    0.4412458 ]
 [ -3.2855194    0.6929818    2.7781472    0.46150094  -0.45086998
    0.65019524]
 [ -4.364477     0.87604487   2.9793143    0.61728126  -0.5937211
    0.86381125]
 [ -5.4284225    1.05996      3.1717136    0.77157027  -0.7301032
    1.0855012 ]
 [ -6.522745     1.2406223    3.3603013    0.93041736  -0.86720294
    1.3115559 ]
 [ -7.630296     1.4210577    3.5460904    1.0925899   -1.0056775
    1.5364754 ]
 [ -8.713226     1.6070437    3.7269533    1.2536659   -1.1422923
    1.7578521 ]
 [ -9.816599     1.797044     3.9023683    1.4152267   -1.2814488
    1.9771683 ]
 [-10.975772     1.9885356    4.0732584    1.5798028   -1.424406
    2.2005541 ]
 [-12.114705     2.1864862    4.239405     1.7431915   -1.5591667
    2.4352508 ]
 [-13.194851     2.3922246    4.400227     1.9034963   -1.6822653
    2.678908  ]
 [-14.302092     2.597243     4.5557265    2.06641     -1.8087406
    2.9195635 ]
 [-15.467853     2.7984676    4.7052474    2.23222     -1.9442332
    3.1522837 ]
 [-16.607662     3.0032256    4.8464       2.392235    -2.0747974
    3.386411  ]
 [-17.724266     3.2114024    4.9749618    2.545282    -2.1966677
    3.6309931 ]
 [-18.900156     3.4145553    5.0896344    2.6989145   -2.3201647
```

```
   3.8837597 ]
 [-20.0862       3.6130273    5.196701     2.8535326   -2.4427636
   4.1367493 ]
 [-21.23248      3.8084521    5.298637     3.0073874   -2.561035
   4.383404  ]
 [-22.44708      3.9939146    5.3856797    3.1640573   -2.6859055
   4.6256094 ]]
```

[13]:
```
np.save("data_test", x)
```

[29]:
```
with open("val.txt",'r') as data_file:
    for line in data_file:
        data = line.split()
        res = [s.strip() for s in data[0].split(',')]
        print(res[0], res[1],res[2],res[3],res[4],res[5])
```

```
0 1 2 3 4 5
-1.1244261 0.3311052 2.3444054 0.13440591 -0.15608305 0.22917116
-2.1868036 0.5127536 2.5655348 0.29968756 -0.30188364 0.4412458
-3.2855194 0.6929818 2.7781472 0.46150094 -0.45086998 0.65019524
-4.364477 0.87604487 2.9793143 0.61728126 -0.5937211 0.86381125
-5.4284225 1.05996 3.1717136 0.77157027 -0.7301032 1.0855012
-6.522745 1.2406223 3.3603013 0.93041736 -0.86720294 1.3115559
-7.630296 1.4210577 3.5460904 1.0925899 -1.0056775 1.5364754
-8.713226 1.6070437 3.7269533 1.2536659 -1.1422923 1.7578521
-9.816599 1.797044 3.9023683 1.4152267 -1.2814488 1.9771683
-10.975772 1.9885356 4.0732584 1.5798028 -1.424406 2.2005541
-12.114705 2.1864862 4.239405 1.7431915 -1.5591667 2.4352508
-13.194851 2.3922246 4.400227 1.9034963 -1.6822653 2.678908
-14.302092 2.597243 4.5557265 2.06641 -1.8087406 2.9195635
-15.467853 2.7984676 4.7052474 2.23222 -1.9442332 3.1522837
-16.607662 3.0032256 4.8464 2.392235 -2.0747974 3.386411
-17.724266 3.2114024 4.9749618 2.545282 -2.1966677 3.6309931
-18.900156 3.4145553 5.0896344 2.6989145 -2.3201647 3.8837597
-20.0862 3.6130273 5.196701 2.8535326 -2.4427636 4.1367493
-21.23248 3.8084521 5.298637 3.0073874 -2.561035 4.383404
-22.44708 3.9939146 5.3856797 3.1640573 -2.6859055 4.6256094
```

[37]:
```
from ikin import *
rbt = Robot()
with open("val.txt",'r') as data_file:
    for line in data_file:
        data = line.split()
        #print(data[0])
        res = [s.strip() for s in data[0].split(',')]
        #print(res[0], res[1],res[2],res[3],res[4],res[5])
        #print(float(res[0]),␣
 ↪float(res[1]),float(res[2]),float(res[3]),float(res[4]),float(res[5]))
```

```
        print(rbt.goto(float(res[0]),␣
↪float(res[1]),float(res[2]),float(res[3]),float(res[4]),float(res[5])))
```

```
[8.826, 3.44, 7.734, -2.517, -4.968, -13.319]
[1.074, 0.01, 0.019, -0.015, -0.007, 0.023]
[2.072, 0.017, 0.009, -0.019, -0.012, 0.024]
[3.067, 0.011, -0.009, -0.013, -0.023, 0.035]
[4.041, -0.002, -0.019, -0.008, -0.028, 0.037]
[5.009, -0.016, -0.017, -0.008, -0.03, 0.024]
[5.996, -0.029, -0.015, -0.009, -0.04, 0.012]
[6.991, -0.032, -0.018, -0.01, -0.052, 0.006]
[7.974, -0.024, -0.026, -0.015, -0.058, 0.004]
[8.962, -0.014, -0.044, -0.016, -0.067, 0.01]
[9.976, -0.009, -0.064, -0.009, -0.086, 0.02]
[10.991, -0.001, -0.068, -0.008, -0.095, 0.014]
[11.987, 0.017, -0.056, -0.018, -0.087, -0.013]
[12.994, 0.04, -0.053, -0.026, -0.087, -0.03]
[14.018, 0.061, -0.069, -0.023, -0.1, -0.023]
[15.023, 0.081, -0.081, -0.021, -0.104, -0.02]
[16.013, 0.089, -0.077, -0.024, -0.1, -0.037]
[17.026, 0.079, -0.07, -0.022, -0.108, -0.056]
[18.043, 0.072, -0.064, -0.021, -0.122, -0.073]
[19.038, 0.079, -0.061, -0.025, -0.132, -0.087]
[20.053, 0.077, -0.07, -0.018, -0.165, -0.087]
```

```
[ ]:
```