neural-network_v0.1

December 6, 2023

- Flowchart
  1. Module import / Define functions
  2. Prepare dataset (training & testing)
  3. Build an NN Model
  4. Train the NN model by fitting training data
  5. Evaluate the quality of NN model
  6. Predict test data
- Reference
  - TensorFlow: https://www.tensorflow.org
  - Keras: https://keras.io/ja/

0.0.1 Version 0.1 (Last update: 2023/12/06)

1 Define functions

[ ]:

```
# --- initialization for generating random numbers

def init_WinOS(iseed):
    # --- clear session
    import keras.backend as K
    K.clear_session()
    # --- set OS environment
    import os
    os.environ["PYTHONHASHSEED"] = str(iseed)
    os.environ["TF_DETERMINISTIC_OPS"] = "true"
    os.environ["TF_CODNN_DETERMINISTIC"] = "true"
    # --- initialization
    np.random.seed(iseed)
    tr.random.set_seed(iseed)


def init_MacOS(iseed):
    # --- initialization
    tf.random.set_seed(iseed)
```

[ ]:

```
# --- Function for plotting training history
def PlotHistory(history, metrics):
    '''
        metrics = {MSE|MAE|...}
    '''

    # --- get epoch
    hist = pd.DataFrame(history.history)
    hist["Epoch"] = history.epoch

    # --- plot figures
    plt.figure()
    plt.xlabel("Number of epochs")
    plt.ylabel(f"{metrics} of {target}")
    plt.plot(hist["Epoch"], hist[metrics],        label="Training")
    plt.plot(hist["Epoch"], hist["val_"+metrics], label="Validation")
#    plt.plot(hist["Epoch"].values, hist[metrics].values,       ␣
↪label="Training")        # for pandas>3.4 maybe
#    plt.plot(hist["Epoch"].values, hist["val_"+metrics].values,␣
↪label="Validation")
    plt.yscale('log')
    plt.legend()
    plt.show()
```

[ ]:

```
# --- Function for plotting actual-predicted plot
def PlotCorrelation(y_train, y_predict):

    # --- plot figures
    plt.axis('equal')
    plt.axis('square')
    plt.xlabel(f"Actual")
    plt.ylabel(f"Predicted")
    plt.scatter(y_train, y_predict, color='blue', alpha=0.3)
    plt.xlim([-1.2, 0])
    plt.ylim([-1.2, 0])
    plt.plot([-100, 100], [-100, 100], color='gray')
    plt.show()
```

2 Module import

[ ]:

```
# --- import modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
import tensorflow as tf
```

```
from tensorflow   import keras
from keras.layers import Dense
```

[ ]:

```
# --- Version information
import platform
import matplotlib
import tensorflow
import sklearn
import keras

ver = "0.0.1"   # version of this program

print(f"Vesion information:")
print(f"  This program : {ver}")
print(f"  Python       : {platform.python_version()}")
print(f"  Pandas       : {pd.__version__}")
print(f"  Numpy        : {np.__version__}")
print(f"  Matplotlib   : {matplotlib.__version__}")
print(f"  TensorFlow   : {tensorflow.__version__}")
print(f"  Scikit-learn : {sklearn.__version__}")
print(f"  Keras        : {keras.__version__}")
```

```
Vesion information:
  This program : 0.0.1
  Python       : 3.8.3
  Pandas       : 1.0.5
  Numpy        : 1.18.5
  Matplotlib   : 3.2.2
  TensorFlow   : 2.6.0
  Scikit-learn : 0.23.1
  Keras        : 2.6.0
```

3 Initialization

[ ]:

```
# --- initialization (only for Windows-OS)
iseed = 1
OS = "Mac"          # Win/Mac

if OS == 'Win':
    init_WinOS(iseed)
elif OS == 'Mac':
    init_MacOS(iseed)
```

4 Prepare dataset (training & testing)

[ ]:

```
# --- read iput data
filename_inp = "magn_CoFe9.csv"
target = "dEform_eV"
df = pd.read_csv(filename_inp)

# --- separate X/Y data
data_x = df[["a1", "a2", "a3", "a4", "a5", "a6", "a7", "a8", "a9"]]
data_y = df[[target]]
train_size = 0.3
train_x, test_x, train_y, test_y = train_test_split(data_x, data_y,␣
 ↪train_size=train_size, shuffle=True, random_state=1)
print(f"Number of training data: {len(train_x):>5}")
print(f"Number of testing  data: {len(test_x):>5}")
```

```
Number of training data:   153
Number of testing  data:   359
```

5 Build an NN model

[ ]:

```
'''
    activation = {relu|linear|sigmoid|...}
    optimizer = {SGD|Adam|...}
'''
print("Building a model ...")

# --- set NN archnitecture
model = keras.Sequential()
model.add(Dense(64, activation="relu", input_shape=[train_x.shape[1]]))
model.add(Dense(64, activation="relu"))
model.add(Dense(32, activation="relu"))
model.add(Dense(16, activation="relu"))
model.add(Dense( 1))

# --- compile the model
model.compile(loss="MSE", optimizer="SGD", metrics=["MSE"])
model.summary()
```

```
Building a model …
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                640
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
```

```
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_3 (Dense)              (None, 16)                528
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 17
=================================================================
Total params: 7,425
Trainable params: 7,425
Non-trainable params: 0
_________________________________________________________________
```

6 Train the NN model by fitting training data

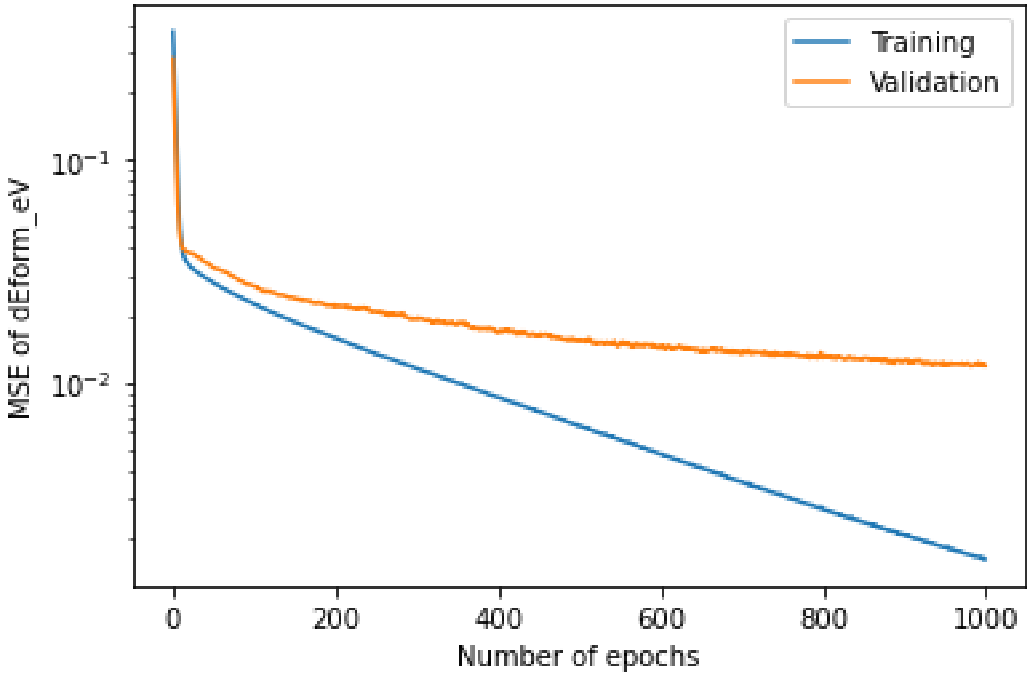
[ ]:

```
print("Fitting the model started ...")
model_history = model.fit(train_x, train_y,
                          epochs = 1000, validation_split = 0.3,
                          verbose = 0,  # <- change "3" to see training progress
                          )
```

```
Fitting the model started …
```

[ ]:

```
# --- plot training history
PlotHistory(model_history, "MSE")
```



7 Evaluate a quality of NN model

[ ]:

```
# --- evaluate the NN model (for training)
result_train = model.evaluate(train_x, train_y, verbose=0)
print("Validation for training data")
print(f"    Loss function       : {result_train[0]: .4E}")

# --- evaluate the NN model (for testing)
result_test = model.evaluate(test_x, test_y, verbose=0)
print("Validation for testing data")
print(f"    Loss function       : {result_test[0]: .4E}")
```

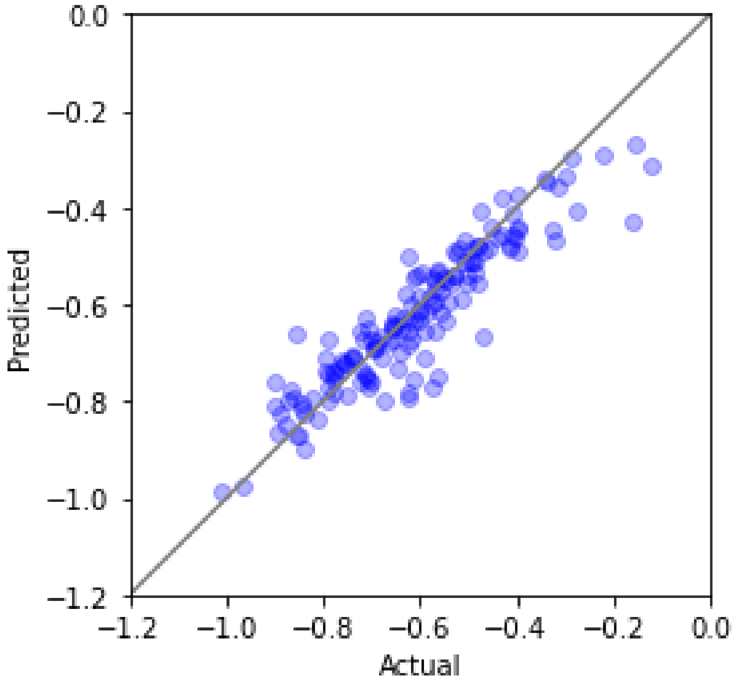
```
Validation for training data
    Loss function       :  4.7103E-03
Validation for testing data
    Loss function       :  1.0297E-02
```

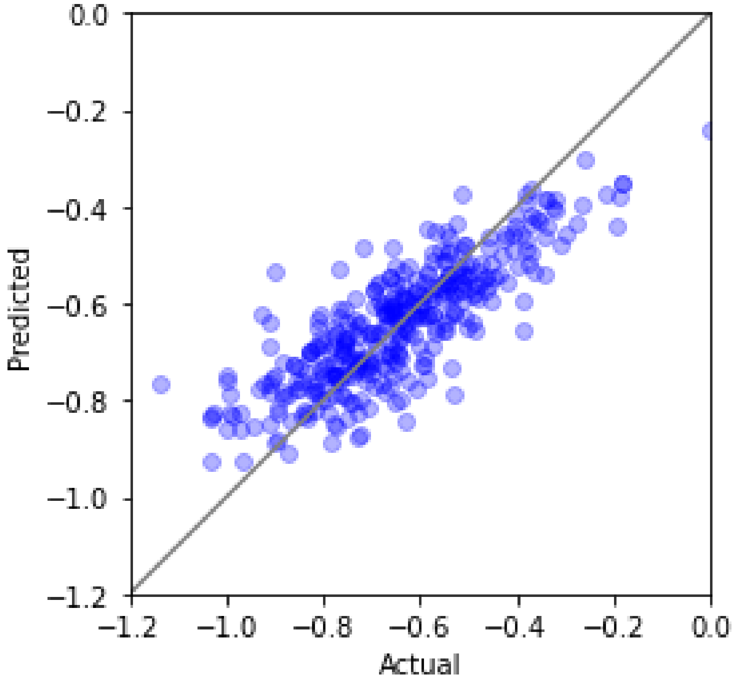
8 Predict test data

[ ]:

```
# --- predict the training/testing model
predict_train = model.predict(train_x).flatten()     # reproducibility
predict_test  = model.predict(test_x).flatten()
# --- Plot actual-predicted
PlotCorrelation(train_y, predict_train)
PlotCorrelation(test_y,  predict_test)

# --- evaluate the R2 score
r2score_train = r2_score(train_y, predict_train)
r2score_test = r2_score(test_y, predict_test)
print(f"R2 score for training data : {r2score_train: .4E}")
print(f"R2 score for testing  data : {r2score_test: .4E}")
```





```
R2 score for training data :  8.3506E-01
R2 score for testing  data :  6.6620E-01
```

[ ]:

```
hist = pd.DataFrame(model_history.history)
hist["Epoch"] = model_history.epoch
print(min(hist["loss"]))
```

```
0.0016073953593149781
```

8.1 Version log

- Version 0.1 (2023/12/06)
  - initialization for random number generator (Win) added.

- Version 0.0 (2022/07/04)

[ ]:

```

```