**Abstract**

In a stepped spillway, the spillway face is provided with a series of steps from near the crest to the toe. The energy dissipation caused by the steps reduces the size of the energy dissipater, generally provided at the toe of the spillway. The steps produce considerable energy dissipation along the spillway and reduce the size of the required downstream energy dissipation basin. The main objective is to estimate the proper values of energy dissipation of skimming flow regime over stepped spillways because of imprecise, insufficient, ambiguous and uncertain data available. The primary focus of this research is to investigate the accuracy of a artificial neural network(ANN) approach for estimating the energy dissipation of the skimming flow regime over stepped spillway because of the imprecise, insufficient, ambiguous and uncertain data available. Artificial neural network (ANN) has applied to estimate the energy dissipation since they are capable of correlating large and complex data sets without any prior knowledge of the relationships among them.

Algorithm:

Back propagation Artificial Neural Network

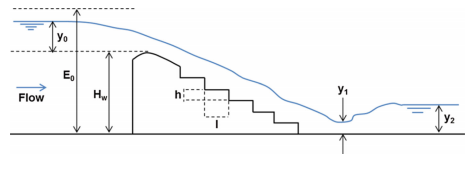
Software Used:

Anaconda/ Spyder/jupyter notebook

Programming Language used for implementation: python

**Feature Selection**

Main involved parameters on energy waste over stepped spillways. [1]



To show the influence of hydraulic parameters and geometrical parameters of steps on performance of stepped spillway to dissipate the energy, involved parameters are collected in below equation.(using Bernoulli equation)

..(1)

Where, q is discharge of flow per width of spillway

l is the length of a step

h is the height of a step

Hw is the height of dam

g is gravitational acceleration

N is number of steps

Using the Buckingham second theory as common approach of dimensional analysis, the most important parameters on energy dissipation are derived in below equation

/gHw3 (2)

Assuming DN= q2/gH3 and S=h/l

(3)

**Dataset Used**

For this project we have used the experimental dataset by Salmasi and O¨ zger[2]. To develop the soft computing techniques for prediction of the energy dissipation of flow the stepped spillways, collected dataset is divided into two groups as training and testing. To assign dataset properly, randomly the dataset is divided.

Sample dataset:-

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Row | OE/E 0 | Fr1 | yc/h | q2/gH 3 | N | S |
| 1 | 61.523 | 4.706 | 0.608 | 0.001105 | 5 | 45 |
| 2 | 60.53 | 4.877 | 0.596 | 0.001044 | 5 | 45 |
| 3 | 63.39 | 4.758 | 0.565 | 0.00089 | 5 | 45 |
| 4 | 65.931 | 4.734 | 0.524 | 0.00071 | 5 | 45 |
| 5 | 70.308 | 4.658 | 0.459 | 0.000476 | 5 | 45 |
| 6 | 78.157 | 3.931 | 0.404 | 0.000325 | 5 | 45 |
| 7 | 70.203 | 6.223 | 0.314 | 0.000153 | 5 | 45 |
| 8 | 84.571 | 4.534 | 0.233 | 6.3E-05 | 5 | 45 |
| 9 | 60.486 | 4.807 | 1.231 | 0.001111 | 10 | 45 |
| 10 | 61.312 | 4.979 | 1.145 | 0.000896 | 10 | 45 |
| 11 | 67.193 | 5.283 | 0.878 | 0.000404 | 10 | 45 |
| 12 | 80.723 | 3.642 | 0.78 | 0.000283 | 10 | 45 |
| 13 | 65.084 | 7.361 | 0.601 | 0.00013 | 10 | 45 |
| 14 | 84.384 | 3.65 | 0.619 | 0.000141 | 10 | 45 |
| 15 | 81.226 | 5.346 | 0.473 | 6.3E-05 | 10 | 45 |
| 16 | 86.788 | 4.085 | 0.454 | 5.6E-05 | 10 | 45 |
| 17 | 87.066 | 4.101 | 0.442 | 5.1E-05 | 10 | 45 |
| 18 | 84.818 | 5.505 | 0.364 | 2.9E-05 | 10 | 45 |
| 19 | 53.397 | 5.346 | 1.956 | 0.001271 | 15 | 45 |
| 20 | 60.26 | 4.82 | 1.875 | 0.001121 | 15 | 45 |
| 21 | 62.144 | 4.81 | 1.779 | 0.000956 | 15 | 45 |
| 22 | 70.89 | 4.417 | 1.477 | 0.000547 | 15 | 45 |
| 23 | 70.062 | 5.166 | 1.242 | 0.000326 | 15 | 45 |
| 24 | 64.96 | 6.336 | 1.124 | 0.000241 | 15 | 45 |
| 25 | 71.521 | 5.883 | 0.991 | 0.000165 | 15 | 45 |

Total number of rows – 154

Total number of columns - 6

**Neural Network Description**

ANN is the most common type of soft computing technique used in most area of science and engineering. The concept of ANN development was given from nerve cells in the human brain. In this idea, each input multiplied with a weight (wij) and passed through an activation function and then summed by a bias(). Mathematical expression of which is given below. In this equation, xi is inputs to each neuron.



Where I is the results of computation of each neuron. Various activation functions have been introduced that are most widely used. This operation is conducted for all neurons considered in the network. A network may contained one or more hidden layer. . When a network has more than one hidden layer, outcomes of previous layer are considered for inputs of next hidden layer. This type of neural network is called multilayer perceptron (MLP). Weights and biases for each neuron existed in network are justified using least square error approach. To this end, Levenberg-Marquardt algorithm(LMA) is common approach. But recent modern optimization techniques such as Genetic algorithms and particle swarm optimization have been applied.

Algorithm

1. Collecting the data and pre-processing the data.

* Import the numpy, matplotlib.pyplot, pandas, keras
* Import the dataset
* Store the features of data in x
* Store the energy dissipated of data in y
* Splitting the dataset into the training set and test set
* Scale the features

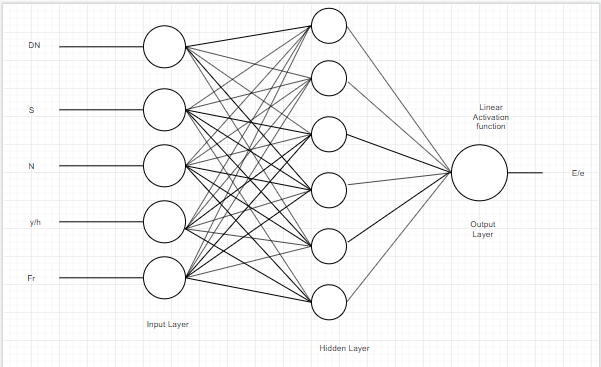
1. Now let’s make the ANN.

* Importing the keras libraries and packages.
* Initialising the sequential ANN classifier.
* Adding the input layer
* Adding the first hidden layer with 6 nodes
* Adding the output layer with one output of energy dissipated prediction
* Compiling the ANN
* Fitting the ANN to the Training set

1. Making predictions and evaluating the model

* Predicting the test set results
* Calculating root mean squares error
* Calculate the accuracy

Architecture-

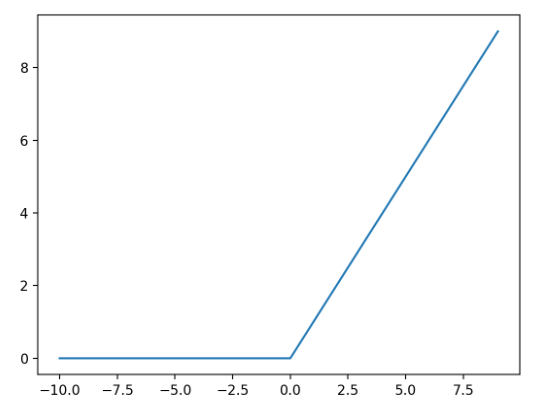


Back propagation Artificial Neural Network

Activation Functions used:-

1. ReLu – A(x) = max(0,x) [used in hidden layer and input layer]

The function is linear for values greater than zero, meaning it has a lot of the desirable properties of a linear activation function when training a neural network using backpropagation. Yet, it is a nonlinear function as negative values are always output as zero.



1. Linear function – y=ax [output layer]

**Python Code**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import keras

#Part1 : Collecting the data and pre-processing the data

# Importing the dataset

dataset = pd.read\_csv('dataset.csv')

#Storing the features of data in x

x = dataset.iloc[:,2:7 ].values

#Storing the energy dispatted(OE/E 0) of data in y

y = dataset.iloc[:,1].values

print(x.shape)

print(y)

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.2, random\_state = 0)

print(x\_train.shape)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train)

x\_test = sc.transform(x\_test)

# Part 2 - Now let's make the ANN!

# Importing the Keras libraries and packages

from keras.models import Sequential

from keras.layers import Dense

# Initialising the ANN

classifier = Sequential()

# Adding the input layer

classifier.add(Dense(output\_dim = 6, kernel\_initializer='normal', activation = 'relu', input\_dim = x\_train.shape[1]))

# Adding the first hidden layer with 6nodes

classifier.add(Dense(output\_dim = 6, kernel\_initializer='normal', activation = 'relu'))

# Adding the output layer with one output of energy dispatted prediction

classifier.add(Dense(output\_dim = 1, kernel\_initializer='normal', activation='linear'))

# Compiling the ANN

classifier.compile(loss='mean\_absolute\_error', optimizer='adam', metrics=['mean\_absolute\_error'])

# Fitting the ANN to the Training set

classifier.fit(x\_train, y\_train, batch\_size = 10, nb\_epoch = 100)

# Part 3 - Making the predictions and evaluating the model

# Predicting the Test set results

y\_pred = classifier.predict(x\_test)

# Calculating Root Mean Squares Error

size=y\_pred.shape[0];

rmse = 0

for i in range(size):

rmse += (y\_test[i] - y\_pred[i]) \*\* 2

rmse = np.sqrt(rmse/size)

print("root mean square error of test set :")

print(rmse)

arr = x\_test[:,0].reshape(31,1)

arr = np.append(arr,y\_pred,axis=1)

arr = np.append(arr,y\_test.reshape(31,1),axis=1)

arr = arr[arr[:,0].argsort()]

plt.plot(arr[:,0],arr[:,2],label="Theoretical")

plt.plot(arr[:,0],arr[:,1],linestyle='--',label="predicted")

plt.ylabel('Predicted vs Theoretical value')

plt.legend()

plt.show()

**Result**

Epoch 1/100

123/123 [==============================] - 1s 7ms/step - loss: 59.7308 - mean\_absolute\_error: 59.7308

Epoch 2/100

123/123 [==============================] - 0s 202us/step - loss: 59.7148 - mean\_absolute\_error: 59.7148

Epoch 3/100

123/123 [==============================] - 0s 255us/step - loss: 59.6952 - mean\_absolute\_error: 59.6952

Epoch 4/100

123/123 [==============================] - 0s 213us/step - loss: 59.6712 - mean\_absolute\_error: 59.6712

Epoch 5/100

123/123 [==============================] - 0s 281us/step - loss: 59.6407 - mean\_absolute\_error: 59.6407

Epoch 6/100

123/123 [==============================] - 0s 214us/step - loss: 59.6010 - mean\_absolute\_error: 59.6010

Epoch 7/100

123/123 [==============================] - 0s 189us/step - loss: 59.5487 - mean\_absolute\_error: 59.5487

Epoch 8/100

123/123 [==============================] - 0s 184us/step - loss: 59.4815 - mean\_absolute\_error: 59.4815

Epoch 9/100

123/123 [==============================] - 0s 217us/step - loss: 59.3965 - mean\_absolute\_error: 59.3965

Epoch 10/100

123/123 [==============================] - 0s 242us/step - loss: 59.2895 - mean\_absolute\_error: 59.2895

Epoch 11/100

123/123 [==============================] - 0s 200us/step - loss: 59.1571 - mean\_absolute\_error: 59.1571

Epoch 12/100

123/123 [==============================] - 0s 232us/step - loss: 58.9940 - mean\_absolute\_error: 58.9940

Epoch 13/100

123/123 [==============================] - 0s 190us/step - loss: 58.7946 - mean\_absolute\_error: 58.7946

Epoch 14/100

123/123 [==============================] - 0s 244us/step - loss: 58.5542 - mean\_absolute\_error: 58.5542

Epoch 15/100

123/123 [==============================] - 0s 189us/step - loss: 58.2728 - mean\_absolute\_error: 58.2728

Epoch 16/100

123/123 [==============================] - 0s 175us/step - loss: 57.9436 - mean\_absolute\_error: 57.9436

Epoch 17/100

123/123 [==============================] - 0s 231us/step - loss: 57.5659 - mean\_absolute\_error: 57.5659

Epoch 18/100

123/123 [==============================] - 0s 248us/step - loss: 57.1253 - mean\_absolute\_error: 57.1253

Epoch 19/100

123/123 [==============================] - 0s 256us/step - loss: 56.6242 - mean\_absolute\_error: 56.6242

Epoch 20/100

123/123 [==============================] - 0s 241us/step - loss: 56.0663 - mean\_absolute\_error: 56.0663

Epoch 21/100

123/123 [==============================] - 0s 249us/step - loss: 55.4331 - mean\_absolute\_error: 55.4331

Epoch 22/100

123/123 [==============================] - 0s 215us/step - loss: 54.7314 - mean\_absolute\_error: 54.7314

Epoch 23/100

123/123 [==============================] - 0s 220us/step - loss: 53.9545 - mean\_absolute\_error: 53.9545

Epoch 24/100

123/123 [==============================] - 0s 252us/step - loss: 53.1024 - mean\_absolute\_error: 53.1024

Epoch 25/100

123/123 [==============================] - 0s 265us/step - loss: 52.1550 - mean\_absolute\_error: 52.1550

Epoch 26/100

123/123 [==============================] - 0s 245us/step - loss: 51.1353 - mean\_absolute\_error: 51.1353

Epoch 27/100

123/123 [==============================] - 0s 271us/step - loss: 50.0216 - mean\_absolute\_error: 50.0216

Epoch 28/100

123/123 [==============================] - 0s 269us/step - loss: 48.8074 - mean\_absolute\_error: 48.8074

Epoch 29/100

123/123 [==============================] - 0s 209us/step - loss: 47.4980 - mean\_absolute\_error: 47.4980

Epoch 30/100

123/123 [==============================] - 0s 187us/step - loss: 46.1460 - mean\_absolute\_error: 46.1460

Epoch 31/100

123/123 [==============================] - 0s 184us/step - loss: 44.7049 - mean\_absolute\_error: 44.7049

Epoch 32/100

123/123 [==============================] - 0s 176us/step - loss: 43.2161 - mean\_absolute\_error: 43.2161

Epoch 33/100

123/123 [==============================] - 0s 215us/step - loss: 41.6310 - mean\_absolute\_error: 41.6310

Epoch 34/100

123/123 [==============================] - 0s 242us/step - loss: 39.9737 - mean\_absolute\_error: 39.9737

Epoch 35/100

123/123 [==============================] - 0s 256us/step - loss: 38.2620 - mean\_absolute\_error: 38.2620

Epoch 36/100

123/123 [==============================] - 0s 226us/step - loss: 36.5353 - mean\_absolute\_error: 36.5353

Epoch 37/100

123/123 [==============================] - 0s 158us/step - loss: 34.8070 - mean\_absolute\_error: 34.8070

Epoch 38/100

123/123 [==============================] - 0s 222us/step - loss: 32.9218 - mean\_absolute\_error: 32.9218

Epoch 39/100

123/123 [==============================] - 0s 177us/step - loss: 31.0415 - mean\_absolute\_error: 31.0415

Epoch 40/100

123/123 [==============================] - 0s 234us/step - loss: 29.0776 - mean\_absolute\_error: 29.0776

Epoch 41/100

123/123 [==============================] - 0s 221us/step - loss: 27.0644 - mean\_absolute\_error: 27.0644

Epoch 42/100

123/123 [==============================] - 0s 308us/step - loss: 25.0817 - mean\_absolute\_error: 25.0817

Epoch 43/100

123/123 [==============================] - 0s 228us/step - loss: 23.0694 - mean\_absolute\_error: 23.0694

Epoch 44/100

123/123 [==============================] - 0s 219us/step - loss: 21.0521 - mean\_absolute\_error: 21.0521

Epoch 45/100

123/123 [==============================] - 0s 241us/step - loss: 19.1515 - mean\_absolute\_error: 19.1515

Epoch 46/100

123/123 [==============================] - 0s 204us/step - loss: 17.4950 - mean\_absolute\_error: 17.4950

Epoch 47/100

123/123 [==============================] - 0s 211us/step - loss: 15.8160 - mean\_absolute\_error: 15.8160

Epoch 48/100

123/123 [==============================] - 0s 216us/step - loss: 14.2568 - mean\_absolute\_error: 14.2568

Epoch 49/100

123/123 [==============================] - 0s 256us/step - loss: 13.0742 - mean\_absolute\_error: 13.0742

Epoch 50/100

123/123 [==============================] - 0s 340us/step - loss: 12.1454 - mean\_absolute\_error: 12.1454

Epoch 51/100

123/123 [==============================] - 0s 191us/step - loss: 11.4386 - mean\_absolute\_error: 11.4386

Epoch 52/100

123/123 [==============================] - 0s 173us/step - loss: 10.8640 - mean\_absolute\_error: 10.8640

Epoch 53/100

123/123 [==============================] - 0s 222us/step - loss: 10.5432 - mean\_absolute\_error: 10.5432

Epoch 54/100

123/123 [==============================] - 0s 193us/step - loss: 10.2470 - mean\_absolute\_error: 10.2470

Epoch 55/100

123/123 [==============================] - 0s 178us/step - loss: 9.9828 - mean\_absolute\_error: 9.9828

Epoch 56/100

123/123 [==============================] - 0s 175us/step - loss: 9.7608 - mean\_absolute\_error: 9.7608

Epoch 57/100

123/123 [==============================] - 0s 200us/step - loss: 9.5364 - mean\_absolute\_error: 9.5364

Epoch 58/100

123/123 [==============================] - 0s 200us/step - loss: 9.3259 - mean\_absolute\_error: 9.3259

Epoch 59/100

123/123 [==============================] - 0s 248us/step - loss: 9.0996 - mean\_absolute\_error: 9.0996

Epoch 60/100

123/123 [==============================] - 0s 168us/step - loss: 8.8794 - mean\_absolute\_error: 8.8794

Epoch 61/100

123/123 [==============================] - 0s 190us/step - loss: 8.6890 - mean\_absolute\_error: 8.6890

Epoch 62/100

123/123 [==============================] - 0s 225us/step - loss: 8.4645 - mean\_absolute\_error: 8.4645

Epoch 63/100

123/123 [==============================] - 0s 176us/step - loss: 8.2787 - mean\_absolute\_error: 8.2787

Epoch 64/100

123/123 [==============================] - 0s 160us/step - loss: 8.0771 - mean\_absolute\_error: 8.0771

Epoch 65/100

123/123 [==============================] - 0s 198us/step - loss: 7.8844 - mean\_absolute\_error: 7.8844

Epoch 66/100

123/123 [==============================] - 0s 179us/step - loss: 7.7056 - mean\_absolute\_error: 7.7056

Epoch 67/100

123/123 [==============================] - 0s 255us/step - loss: 7.5490 - mean\_absolute\_error: 7.5490

Epoch 68/100

123/123 [==============================] - 0s 178us/step - loss: 7.3935 - mean\_absolute\_error: 7.3935

Epoch 69/100

123/123 [==============================] - 0s 171us/step - loss: 7.2375 - mean\_absolute\_error: 7.2375

Epoch 70/100

123/123 [==============================] - 0s 205us/step - loss: 7.0977 - mean\_absolute\_error: 7.0977

Epoch 71/100

123/123 [==============================] - 0s 181us/step - loss: 6.9596 - mean\_absolute\_error: 6.9596

Epoch 72/100

123/123 [==============================] - 0s 179us/step - loss: 6.8205 - mean\_absolute\_error: 6.8205

Epoch 73/100

123/123 [==============================] - 0s 200us/step - loss: 6.6902 - mean\_absolute\_error: 6.6902

Epoch 74/100

123/123 [==============================] - 0s 190us/step - loss: 6.5788 - mean\_absolute\_error: 6.5788

Epoch 75/100

123/123 [==============================] - 0s 217us/step - loss: 6.4205 - mean\_absolute\_error: 6.4205

Epoch 76/100

123/123 [==============================] - 0s 204us/step - loss: 6.3435 - mean\_absolute\_error: 6.3435

Epoch 77/100

123/123 [==============================] - 0s 181us/step - loss: 6.2602 - mean\_absolute\_error: 6.2602

Epoch 78/100

123/123 [==============================] - 0s 187us/step - loss: 6.1832 - mean\_absolute\_error: 6.1832

Epoch 79/100

123/123 [==============================] - 0s 162us/step - loss: 6.1058 - mean\_absolute\_error: 6.1058

Epoch 80/100

123/123 [==============================] - 0s 230us/step - loss: 6.0722 - mean\_absolute\_error: 6.0722

Epoch 81/100

123/123 [==============================] - 0s 169us/step - loss: 5.9912 - mean\_absolute\_error: 5.9912

Epoch 82/100

123/123 [==============================] - 0s 147us/step - loss: 5.9233 - mean\_absolute\_error: 5.9233

Epoch 83/100

123/123 [==============================] - 0s 218us/step - loss: 5.8823 - mean\_absolute\_error: 5.8823

Epoch 84/100

123/123 [==============================] - 0s 243us/step - loss: 5.8083 - mean\_absolute\_error: 5.8083

Epoch 85/100

123/123 [==============================] - 0s 228us/step - loss: 5.7745 - mean\_absolute\_error: 5.7745

Epoch 86/100

123/123 [==============================] - 0s 273us/step - loss: 5.6878 - mean\_absolute\_error: 5.6878

Epoch 87/100

123/123 [==============================] - 0s 187us/step - loss: 5.6683 - mean\_absolute\_error: 5.6683

Epoch 88/100

123/123 [==============================] - 0s 223us/step - loss: 5.5965 - mean\_absolute\_error: 5.5965

Epoch 89/100

123/123 [==============================] - 0s 246us/step - loss: 5.5454 - mean\_absolute\_error: 5.5454

Epoch 90/100

123/123 [==============================] - 0s 222us/step - loss: 5.4941 - mean\_absolute\_error: 5.4941

Epoch 91/100

123/123 [==============================] - 0s 262us/step - loss: 5.4644 - mean\_absolute\_error: 5.4644

Epoch 92/100

123/123 [==============================] - 0s 200us/step - loss: 5.4098 - mean\_absolute\_error: 5.4098

Epoch 93/100

123/123 [==============================] - 0s 205us/step - loss: 5.3724 - mean\_absolute\_error: 5.3724

Epoch 94/100

123/123 [==============================] - 0s 232us/step - loss: 5.3596 - mean\_absolute\_error: 5.3596

Epoch 95/100

123/123 [==============================] - 0s 189us/step - loss: 5.3042 - mean\_absolute\_error: 5.3042

Epoch 96/100

123/123 [==============================] - 0s 208us/step - loss: 5.2687 - mean\_absolute\_error: 5.2687

Epoch 97/100

123/123 [==============================] - 0s 154us/step - loss: 5.2333 - mean\_absolute\_error: 5.2333

Epoch 98/100

123/123 [==============================] - 0s 195us/step - loss: 5.1954 - mean\_absolute\_error: 5.1954

Epoch 99/100

123/123 [==============================] - 0s 201us/step - loss: 5.1721 - mean\_absolute\_error: 5.1721

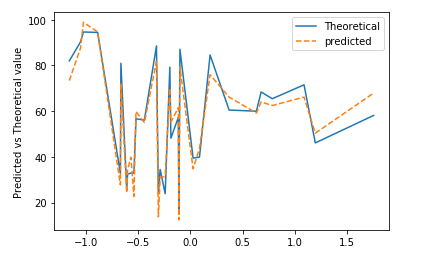
Epoch 100/100

123/123 [==============================] - 0s 211us/step - loss: 5.1457 - mean\_absolute\_error: 5.1457

root mean square error of test set :

[6.3847427]

Accuracy : [93.616]



**References**

[1] Parsaie, A., Haghiabi, A. H., Saneie, M., & Torabi, H. (2016). *Applications of soft computing techniques for prediction of energy dissipation on stepped spillways. Neural Computing and Applications, 29(12), 1393–1409.* doi:10.1007/s00521-016-2667-z

[2]Salmasi F, O¨ zger M (2014) Neuro-fuzzy approach for estimating energy dissipation in skimming flow over stepped spillways. Arab J Sci Eng 39(8):6099–6108