Configuration Manual

MSc Research Project
Cloud Computing

Sumedh Gursale
Student ID: x18208592

School of Computing
National College of Ireland

Supervisor: Manuel Tova-Izquierdo

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sumedh Gursale
Student ID:	x18208592
Programme:	MSc Cloud Computing
Year:	2019-2020
Module:	MSc Research Project
Supervisor:	Manuel Tova-Izquierdo
Submission Due Date:	17/08/2020
Project Title:	Configuration Manual
Word Count:	711
Page Count:	8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

Signature:	
Date:	17th August 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sumedh Gursale
x18208592

1 Introduction

This configuration manual helps readers to understand the system requirements, setup, specification of the software, hardware used for the research. It also includes detailed explanation of required steps need to follow to implement research project: A Proactive Mechanism To Improve Workload Prediction For Cloud Services Using Machine Learning.

2 System Configuration

2.1 Hardware Specification

- Model: HP Pavilion x360 Convertible 14-dg0xxx
- Processor: Intel(R) Core(TM) i5-8265U CPU @1.60GHz 1.80 GHz
- Operating System: Windows 10
- RAM: 8.00 GB (7.83 GB usable)
- Hard Disk: 256 GB

3 Software Used

3.1 Python Installation

To run the proposed model, to perform necessary operations and get the results Python Software is used. Downloaded python from https://www.python.org/downloads/

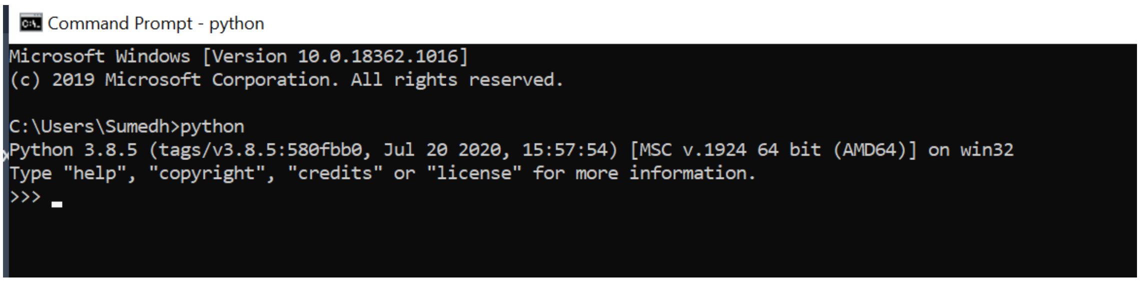

Figure 1: Python Version

3.2 Loading python libraries

The figure below shows the libraries used in this research. To install all required libraries please refer below commands.

python -m pip install –upgrade pip
python -m pip install tensorflow
python -m pip install matplotlib
python -m pip install numpy
python -m pip install sklearn
python -m pip install PyWavelets

matplotlib library is used to plot the curves. sklearn is used to model svr algorithm. tensorflow keras is used to model ANN algorithm and PyWavelets library is used to perform wavelet transformation.

```

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from data_preprocess import *
import pickle
from statistics import mean
import math
import pywt

```

Figure 2: Python Libraries Used

4 Data Generation and Pre-processing

4.1 Data Generation

For this research we have created synthetic data by combining periodic wave functions and pseudo randomness. This data generation system can be tweaked to model any kind of situation, high/low randomness, vary the period etc. File data_generator.py is used to generate the load signal. This generated load signal represents the incoming load on the system and this load can be anything ranging from user requests, processing load etc.

```python
import math
import random
from statistics import mean
import matplotlib.pyplot as plt
import pickle

class CurveGenerator:
    def __init__(self):
        self.y = list()

    #generate the sample data using a combination of periodic functions
    def generate(self,N):
        angle1 = 0
        angle2 = 0
        inc1 = math.pi/180
        inc2 = math.pi/30
        for i in range(N):
            #calculate the functions
            w1 = math.sin(angle1)
            w2 = math.sin(angle2)
            #wave formula
            value = abs(w1+(w2*0.3)*random.random())#------------------------------->change here to customize the curve
            self.y.append ((value))
            #increment the angles
            angle1 += inc1
            angle2 += inc2

    #visualize the data generated by the system
    def plot(self):
        plt.title('input signal')
        x = range(len(self.y))
        y = self.y
        plt.plot(x,y,label='the time series data generated')
        plt.legend()
        plt.xlabel('time in hours')
        plt.ylabel('normalized cpu power')
        plt.show()

    #store the data generated by the system inside a pickle file
    def store(self):
        file = open('./raw_data/generate.pkl','wb')
        pickle.dump(self.y,file)
        file.close()

    def load(self):
        file = open('./raw_data/generate.pkl','rb')
        pickle.load(file)
        file.close()

##################### test/driver code ################################
cg = CurveGenerator()
cg.generate(40000)
#cg.plot()
```

Figure 3: Data Generator

4.2 Data Loading and Pre-processing

For Data loading process, load.py is responsible for loading, generated data and splitting it into rows of 100 values. In pre-processing using data_preprocess.py, data is loaded and divided time series into chunks of 100 values and dividing the 100 values in 90 inputs and 10 output. This stage is pre-processing without wavelet transformation.

```python
####### load the time series data ##########
file = open('./raw_data/generate.pkl','rb')
data = pickle.load(file)
data = np.array(data)
#print(type(data))

###### create the dataset from the time series data ################
# convert the time series data into small portions of 100 values
def create_dataset(data):
    rows = list()
    for i in range(0,len(data)-100,10):
        rows.append(data[i:i+100])
    return np.array(rows)
rows = create_dataset(data)

########################### test #######################
print(rows.shape)
```

Figure 4: Data Loading

```python
# split the row into 2 parts first 90 values used as input and rest 10 values as forecase
X = rows[:,:90]
Y = rows[:,90:]
print(X.shape)

#split the data into training(900) and testing sets(90)
x_train = X[:3000]
x_test = X[3000:]

y_train = Y[:3000]
y_test = Y[3000:]

#################### test code ###################
```

Figure 5: Data pre-processing

5 Proposed Model Implementation

5.1 Wavelet Transformation

In this step the input signal is divided into 2 components cA and cD using discrete wavelet transformation. cA represents low frequency components where as cD represents high frequency components. DWT Haar is the simple way of implementing WT and we have used the same in the process of wavelet transformation. wave_transform.py this is the class to encapsulate the splitting operation. data_process_with_wt.py this module is responsible for the data pre-processing with the discrete wave transformation.

```python
class WaveSpliter:
    #split the original signal into 2 components
    def split(self,signal):
        cA,cD = pywt.dwt(signal,'haar')
        return cA,cD

    #join the 2 components of the system in order to re-construct the original wave
    def join(self,cA,cD):
        y = pywt.idwt(cA,cD,'haar')
        return y
```

Figure 6: Wavelet Transformation

```python
# wave transform
cA_list = list()
cD_list = list()
ws = WaveSpliter()
for row in rows:
    cA,cD = ws.split(row)
    cA_list.append(cA)
    cD_list.append(cD)

#convert the cA into inputs(45) and outputs(5)
cA = np.array(cA_list)
cA_X = cA[:,:45]
cA_Y = cA[:,45:]

#split cA data into training(900) and testing sets(90)
ca_x_train = cA_X[:3000]
ca_x_test = cA_X[3000:]

ca_y_train = cA_Y[:3000]
ca_y_test = cA_Y[3000:]

#convert the cD into inputs(45) and outputs(5)
cD = np.array(cD_list)
cD_X = cD[:,:45]
cD_Y = cD[:,45:]

#split cA data into training(900) and testing sets(90)
cd_x_train = cD_X[:3000]
cd_x_test = cD_X[3000:]

cd_y_train = cD_Y[:3000]
cd_y_test = cD_Y[3000:]

######################### testing ########################
print(cd_x_test.shape)
```

Figure 7: Data processing with Wavelet Transformation

5.2 Combining SVR + ANN

In this step, svr model is used to predict the low frequency variations in the system. cA is applied to SVR algorithm. SVR kernel 'rbf' is used and it is trained on the cA component of the wave. SVR alorithm is defined in composite_svr.py.

```python
############################ define the model ########################################
model_svr = make_pipeline(StandardScaler(), SVR(kernel='rbf',C=10.0, epsilon=0.01))

def predict(x_test):
    predictions = list()
    for x in x_test:
        train = np.array([i for i in range(len(x))]).reshape(-1,1)
        model_svr.fit(train,x)
        test = np.array([i for i in range(len(x),len(x)+5)]).reshape(-1,1)
        #get forecast from the model
        prediction = list(model_svr.predict(test))
        predictions.append(prediction)
    return np.array(predictions)
```

Figure 8: SVR module

In this step, ANN model is used to predict the low frequency variations in the system. cD is applied to SVR algorithm. ADAM optimiser is used in this algorithm. Created the ANN model using tensorflow 2x keras library and it is trained on cD component of the wave. ANN algorithm is defined in composite_ann.py.

```python
model = tf.keras.Sequential()
inputs = keras.Input(shape=(45,), name='digits')

x = layers.Dense(16, activation='relu', name='dense_1')(inputs)
x = layers.Dense(8, activation='relu', name='dense_2')(x)

outputs = layers.Dense(5, name='predictions')(x)
model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss=tf.keras.losses.mse,)

history = model.fit(cd_x_train, cd_y_train,validation_split=0.1,
                    batch_size=16,
                    epochs=20)
'''
```

Figure 9: ANN module

5.3 Inverse Wavelet Transformation

To reconstruct the predicted signal to the original signal this step is followed. reconstruction is done in wave_transform.py.

```python
class WaveSpliter:
    #split the original signal into 2 components
    def split(self,signal):
        cA,cD = pywt.dwt(signal,'haar')
        return cA,cD

    #join the 2 components of the system in order to re-construct the original wave
    def join(self,cA,cD):
        y = pywt.idwt(cA,cD,'haar')
        return y
```

Figure 10: ANN module

5.4 Results

Prediction is done on split data after Wavelet transformation and MSE and RMSE is calculate using predicting and test data that we have. Refer benchmark_composite.py where we have calculated results and plotted original and prediction curve.

```
#################### combine the wavelets ################
#combine the prediction with input to get cD component for inverse wave transformation
cA_predictions = predict(ca_x_test)
cA_combine = np.concatenate((ca_x_test,cA_predictions),axis=1)

cD_predictions = model.predict(cd_x_test)
cD_combine = np.concatenate((cd_x_test,cD_predictions),axis=1)

#########combine the components into single wave
ws = WaveSpliter()
wave_list = list()
for i in range(cD_combine.shape[0]):
    wave = ws.join(cA_combine[i],cD_combine[i])
    wave_list.append(wave)
wave_list = np.array(wave_list)

final_pred = wave_list[:,-10:]

error= final_pred - y_test
error = np.square(error)
mse = np.mean(error)
rmse = np.sqrt(mse)
print('mse of composite algorithm is:',mse)
print('rmse of composite algorithm is:',rmse)

#plot a curve for svr for first 200 values
#as plot more values will clutter the graph
predictions=final_pred.reshape([-1])[:200]
real_values = y_test.reshape([-1])[:200]

plt.title('predicted signal')
x = range(len(predictions))
y1 = predictions
y2 = real_values
plt.plot(x,y1,label='predicted/forecast')
plt.plot(x,y2,label='real values')
plt.legend()
plt.xlabel('time in hours')
plt.ylabel('normalized cpu power')
plt.show()
```

Figure 11: SVR + ANN Results

6 Simple SVR Model

The generated data after loading and pre-processing without applying to the wavelet transformation phase, directly being used and applied to the simple svr module. Predicted values using original input signal and calculated MSE and RMSE is calculated using prediction and test data that we have used. Refer benchmark_svr.py where we have calculated results and plotted original and prediction curve.

```
#loop through the test data and find the rmse error
predictions = predict(x_test)
error = np.square(predictions - y_test)
mse = np.mean(error)
rmse = np.sqrt(mse)
print('mse of svr algorithm is:',mse)
print('rmse of svr algorithm is:',rmse)

#plot a curve for svr for first 200 values
#as plot more values will clutter the graph
predictions=predictions.reshape([-1])[:200]
real_values = y_test.reshape([-1])[:200]

plt.title('predicted signal')
x = range(len(predictions))
y1 = predictions
y2 = real_values
plt.plot(x,y1,label='predicted/forecast')
plt.plot(x,y2,label='real values')
plt.legend()
plt.xlabel('time in hours')
plt.ylabel('normalized cpu power')
plt.show()
```

Figure 12: Simple SVR model Results

References