Time series Thailand Temperature prediction due to Climate Change

Deep Learning Assignment 3

Highlights

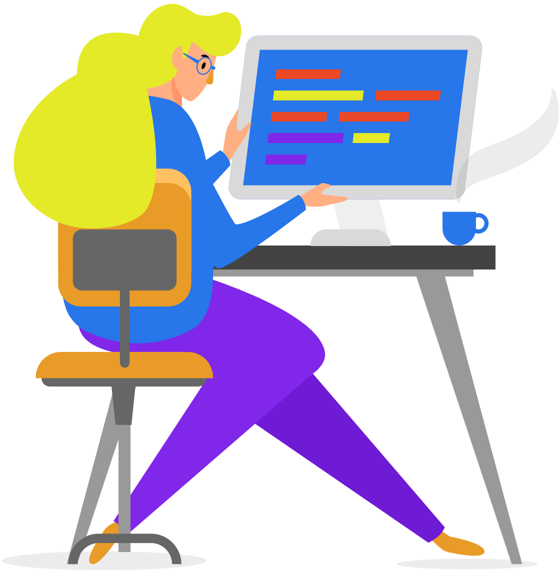
- This project compares three different RNN models (SimpleRNN, LSTM, GRU) with a non-Dl technique model (AutoRegressive) for forecasting the tough temperatures in Thailand

- This study used a kaggle dataset to analyze how average temperatures in Thailand changed rapidly during Jan 1863 - Aug 2013.

- In terms of MAE comparison, you will see what the performance would be for various results of RNN with a non-Dl technique model (Auto-Regressive).

- In this investigation, the interesting finding was that GRU model was the most potent model.

- LSTM and GRU train much faster than SimpleRNN



Agenda

01 Introduction

02 Data Retrieval and Preparation

03 Network architecture and Training of DL method

04 Autoregressive model

05 Results

06 Discussion and Conclusion

Introduction

Some argue that global warming is the greatest peril our generation, while others argue that it is a fiction based on dubious science. In this study, we provide some data so that you can make your own assumptions.

There's a lot of data cleaning and preparation that goes into putting together a long-term study of weather patterns, even more than with other data sets that Kaggle has covered. Technicians used mercury thermometers to collect early data, and any fluctuation in visit duration had an impact on measurements.

Introduction

Many weather stations were relocated in the 1940s due to airport building. Electronic thermometers, which are thought to have a cooling bias, were particularly popular.

Given this intricacy, a multitude of different organizations gather information on climate trends. NOAA's MLOST, NASA's GISTEMP, and the UK's HadCrut are the three most widely used atmospheric and oceanic temperature data sets.

We repackaged the data from a more recent Berkeley Earth compilation, which is linked with Lawrence Berkeley National Laboratory. The Berkeley Earth Surface Temperature Study brings together 1.6 billion temperature reports from 16 repositories.

Introduction

It comes in a neat bundle and can be sliced into interesting subgroups (for example by country). They make the underlying data as well as the code for the alterations they performed public. They also employ approaches that allow for the inclusion of weather records from shorter time series, resulting in fewer observations being discarded.

The average land temperature in this dataset from Kaggle begins in 1750, while the maximum and minimum land temperatures, as well as worldwide ocean and land temperatures, commence in 1850.

Since we live in Thailand, we chose Thailand to focus on the small scale for this scope of study.

Purpose

The aims of this study are:

- To test the accuracy of average Thailand's temperature forecasts using four alternative models. *
- To compare the efficacy of four different model forecasts for average Thailand’s temperatures.

*** The four models in this study are:**
1) RNN 2) LSTM 3) GRU 4) AutoRegressive

Data Retrieval

Climate Change: Earth Surface Temperature Data

Exploring global temperatures since 1750

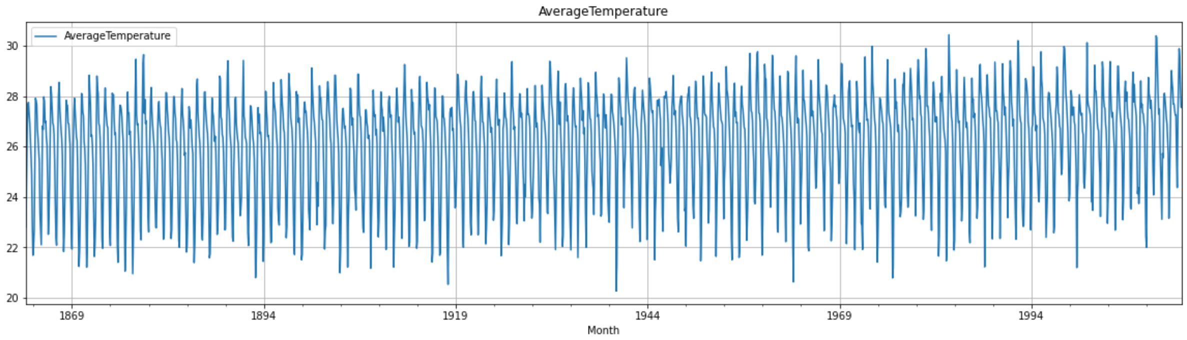
Data Code (558) Discussion (9) Metadata

About Dataset

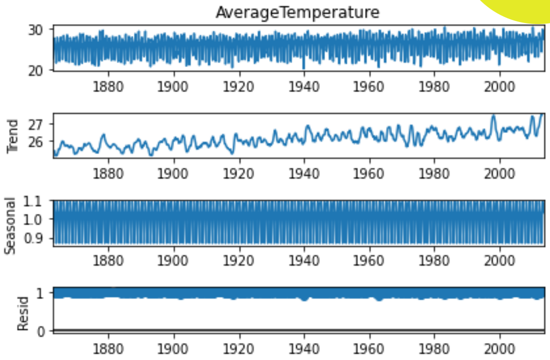
Usability 7.65

- The dataset was mainly retrieved from kaggle dataset- Climate Change: Earth Surface Temperature Data,the open dataset and free for use.
  URL: https://www.kaggle.com/datasets/berkeleyearth/climate-change-earth-surface-temperature-data
- “GlobalLandTemperaturesByCountry” was a csv file to employ in this study.
- There were 1808 rows × 4 columns during Jan 1863 - Aug 2013.

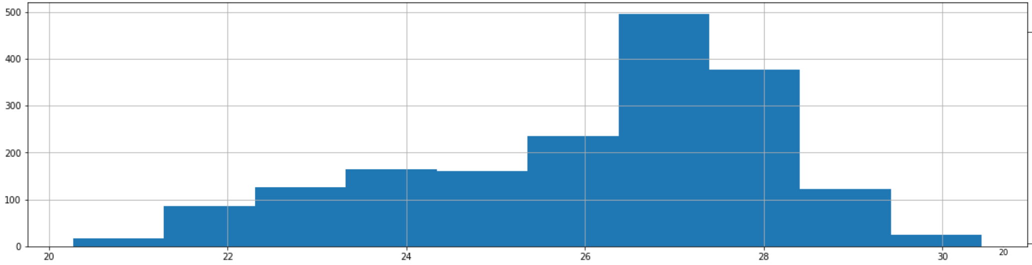
Data Exploration



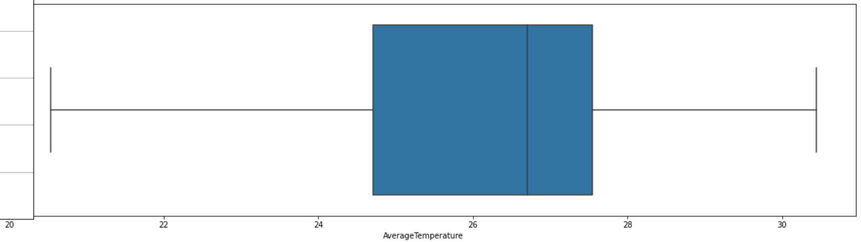
Plot time series data



Multiplicative seasonal decomposition



Histogram plot



Box plot and interquartile range

Data Preparation

	dt	AverageTemperature	AverageTemperatureUncertainty	Country
0	1743-11-01	4.384	2.294	Åland
1	1743-12-01	NaN	NaN	Åland
2	1744-01-01	NaN	NaN	Åland
3	1744-02-01	NaN	NaN	Åland
4	1744-03-01	NaN	NaN	Åland
...	...	...	...	...
577457	2013-05-01	19.059	1.022	Zimbabwe
577458	2013-06-01	17.613	0.473	Zimbabwe
577459	2013-07-01	17.000	0.453	Zimbabwe
577460	2013-08-01	19.759	0.717	Zimbabwe
577461	2013-09-01	NaN	NaN	Zimbabwe

577462 rows × 4 columns

```
Unique country names in the file: 243
```

Original data

	dt	AverageTemperature	AverageTemperatureUncertainty	Country
0	1816-03-01	25.959	1.751	Thailand
1	1816-04-01	27.263	2.960	Thailand
2	1816-05-01	27.932	1.923	Thailand
3	1816-06-01	26.500	1.940	Thailand
4	1816-07-01	25.092	1.605	Thailand
...	...	...	...	...
2366	2013-05-01	29.548	0.286	Thailand
2367	2013-06-01	28.325	0.207	Thailand
2368	2013-07-01	27.564	0.318	Thailand
2369	2013-08-01	27.548	0.289	Thailand
2370	2013-09-01	NaN	NaN	Thailand

2371 rows × 4 columns

```
#Row contain NaN data in each column
dt                               0
AverageTemperature             125
AverageTemperatureUncertainty  125
Country                          0
dtype: int64
```

Thailand data

- The original file contains temperature data for 243 unique countries
- Since our project focuses only on Thailand, the other countries data were filtered out using pandas
- However Thailand data still contains 125 rows of NaN data

Data Preparation

	dt	AverageTemperature	AverageTemperatureUncertainty	Country
46	1820-01-01	NaN	NaN	Thailand
93	1823-12-01	NaN	NaN	Thailand
94	1824-01-01	NaN	NaN	Thailand
95	1824-02-01	NaN	NaN	Thailand
96	1824-03-01	NaN	NaN	Thailand
...	...	...	...	...
558	1862-09-01	NaN	NaN	Thailand
559	1862-10-01	NaN	NaN	Thailand
560	1862-11-01	NaN	NaN	Thailand
561	1862-12-01	NaN	NaN	Thailand
2370	2013-09-01	NaN	NaN	Thailand

Thailand NaN data

	dt	AverageTemperature	AverageTemperatureUncertainty	Country
0	1863-01-01	22.806	2.022	Thailand
1	1863-02-01	24.700	2.396	Thailand
2	1863-03-01	26.599	0.854	Thailand
3	1863-04-01	27.646	1.523	Thailand
4	1863-05-01	27.756	1.296	Thailand
...	...	...	...	...
1803	2013-04-01	29.885	0.234	Thailand
1804	2013-05-01	29.548	0.286	Thailand
1805	2013-06-01	28.325	0.207	Thailand
1806	2013-07-01	27.564	0.318	Thailand
1807	2013-08-01	27.548	0.289	Thailand

Thailand data after cleaning

- After exploring the NaN data, we found that the data after index 561 (dt=1862-12-01) did not contain any NaN data except the last row data
- So the data used will be the data from index 562 to 2369 (the row before the last row)

Data Preparation Code

```
df = pd.read_csv('archive/GlobalLandTemperaturesByCountry.csv')
display(df)
#https://www.kaggle.com/datasets/berkeleyearth/climate-change-earth\-surface-temperature-data?select=GlobalTemperatures.csv
#LandAverageTemperature: global average land temperature in celsius
print('Unique country names in the file:', df['Country'].nunique())
##########################################################
df = df[df['Country']=='Thailand']
df = df.reset_index(drop=True)
display(df)
##########################################################
print('#Row contain NaN data in each colunm')
print(df.isnull().sum())
display(df[df['AverageTemperature'].isnull()])
display(df[df['AverageTemperatureUncertainty'].isnull()])
##########################################################
#^uncomment the code above to see NaN data start from index 46 (1820-01-01) to 561 (1862-12-01) and 2370 (2013-09-01)
df = df[562:-1] #so we use data from index 562 to 2369
df = df.reset_index(drop=True)
print(df.isnull().sum())
df
```

Data Preparation

```
column_data = df['AverageTemperature']

def convertToMatrix(data, feature_timestep):
    x, y = [], []
    for i in range(len(df)-feature_timestep):
        d = i+feature_timestep
        x.append(data[i:d])
        y.append(data[d])
    return np.array(x), np.array(y)
```

CODE: Transform raw data to training data set form

- Raw data will be transformed to deep learning desired form by calling **convertToMatrix** function
- Transformed shape depends on the number of feature timestep parameter called during each experiment

Data Preparation

```
feature_timestep = 4
x, y = convertToMatrix(column_data, feature_timestep)


print(f'{feature_timestep} Features')
print(f'Data Length for {feature_timestep} Features: {len(x)}')
print(f'Length x=y: {len(x)==len(y)}')
print(f'{feature_timestep} Features x shape: {x.shape}')
print(f'{feature_timestep} Features y shape: {y.shape}')
```

```
4 Features
Data Length for 4 Features: 1804
Length x=y: True
4 Features x shape: (1804, 4)
4 Features y shape: (1804,)
```

```
percent_trainset = 0.8
percent_testset = 0.1

rng = np.random.RandomState(random_state_seed)
rng.shuffle(x)
rng = np.random.RandomState(random_state_seed)
rng.shuffle(y)

n_train = int(len(x)*percent_trainset)
n_test = int(len(x)*percent_testset)
n_val = len(x)-n_train-n_test

x_train, y_train = x[:n_train], y[:n_train]
x_val, y_val = x[n_train:n_train+n_val], y[n_train:n_train+n_val]
x_test, y_test = x[n_train+n_val:], y[n_train+n_val:]

print('Split Train/Val/Test')
print(f'{feature_timestep} Features Trainset Shape:',x_train.shape, y_train.shape)
print(f'{feature_timestep} Features Valset Shape:',x_val.shape, y_val.shape)
print(f'{feature_timestep} Features Testset Shape:',x_test.shape, y_test.shape)
```

```
Split Train/Val/Test
4 Features Trainset Shape: (1443, 4) (1443,)
4 Features Valset Shape: (181, 4) (181,)
4 Features Testset Shape: (180, 4) (180,)
```

Data Preparation : Normalized all data by x_train

```
#normalized all data by x_train
minmax_norm = MinMaxScaler().fit(x_train.reshape(-1,1))

x_train_norm = minmax_norm.transform(x_train.reshape(-1,1)).reshape(-1,feature_timestep)
x_val_norm = minmax_norm.transform(x_val.reshape(-1,1)).reshape(-1,feature_timestep)
x_test_norm = minmax_norm.transform(x_test.reshape(-1,1)).reshape(-1,feature_timestep)

y_train_norm = minmax_norm.transform(y_train.reshape(-1,1)).reshape(-1,1)
y_val_norm = minmax_norm.transform(y_val.reshape(-1,1)).reshape(-1,1)
y_test_norm = minmax_norm.transform(y_test.reshape(-1,1)).reshape(-1,1)

print('x shape before newaxis')
print(x_train_norm.shape)
print(x_val_norm.shape)
print(x_test_norm.shape)

#add new axis
x_train_norm = x_train_norm[..., np.newaxis]
'''need input as [[],
               [],
               [],
               []] shape=(4,1)
'''
x_val_norm = x_val_norm[..., np.newaxis]
x_test_norm = x_test_norm[..., np.newaxis]
```

```
print('x shape after newaxis')
print(x_train_norm.shape) #Final input shape must be (n_sample, n_sequence, n_feature per sequence) https://www.youtube.com/watch?v=EnuAP1ZQb4s
print(x_val_norm.shape)
print(x_test_norm.shape)
print('y shape')
print(y_train_norm.shape)
print(y_val_norm.shape)
print(y_test_norm.shape)

#transform to float32
x_train_norm = x_train_norm.astype(np.float32)
x_val_norm = x_val_norm.astype(np.float32)
x_test_norm = x_test_norm.astype(np.float32)

y_train_norm = y_train_norm.astype(np.float32)
y_val_norm = y_val_norm.astype(np.float32)
y_test_norm = y_test_norm.astype(np.float32)
```

```
x shape before newaxis
(1443, 4)
(181, 4)
(180, 4)
x shape after newaxis
(1443, 4, 1)
(181, 4, 1)
(180, 4, 1)
y shape
(1443, 1)
(181, 1)
(180, 1)
```

Network Architecture: Deep RNN

Deep RNN model consists of

- ❑ **SimpleRNN** layer (**128** neurons, activation=**tanh**)
- ❑ **Dropout** layer (**0.5** dropout rate)

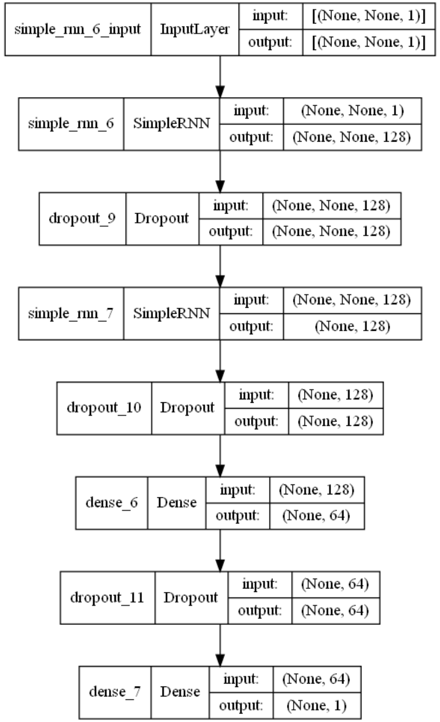
- ❑ **SimpleRNN** layer (**128** neurons, activation=**tanh**)
- ❑ **Dropout** layer (**0.5** dropout rate)

- ❑ **Dense** layer (**64** neurons)
- ❑ **Dropout** layer (**0.5** dropout rate)

- ❑ **Dense** layer (**1** neuron)

```
#deep rnn model
model = Sequential()

model.add(SimpleRNN(128, input_shape=(None,1), return_sequences=True))
model.add(Dropout(0.5))

model.add(SimpleRNN(128))
model.add(Dropout(0.5))

model.add(Dense(64))
model.add(Dropout(0.5))

model.add(Dense(1))
```

CODE: Deep RNN model



Deep RNN model structure

Network Architecture: LSTM

Deep RNN model consists of

- ❑ **LSTM** layer (**128** neurons, activation=**tanh**)
- ❑ **Dropout** layer (**0.5** dropout rate)

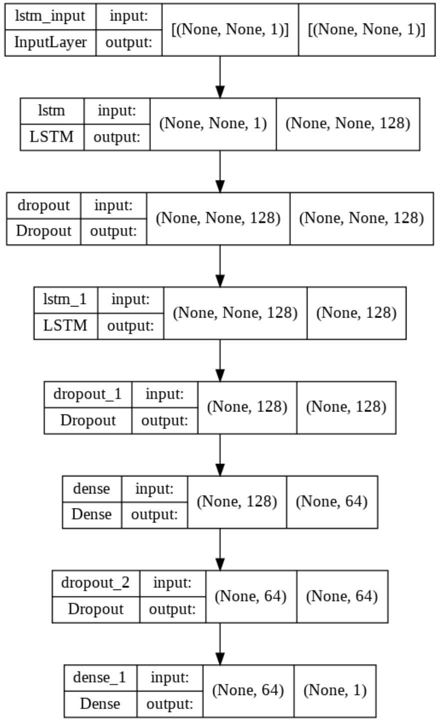
- ❑ **LSTM** layer (**128** neurons, activation=**tanh**)
- ❑ **Dropout** layer (**0.5** dropout rate)

- ❑ **Dense** layer (**64** neurons)
- ❑ **Dropout** layer (**0.5** dropout rate)

- ❑ **Dense** layer (**1** neuron)

```python
#deep LSTM model
model = Sequential()

model.add(LSTM(128, input_shape=(None,1), return_sequences=True))
model.add(Dropout(0.5))

model.add(LSTM(128))
model.add(Dropout(0.5))

model.add(Dense(64))
model.add(Dropout(0.5))

model.add(Dense(1))
```

CODE: LSTM model



LSTM model structure

Network Architecture: GRU

Deep RNN model consists of

- ❑ **GRU** layer (**128** neurons, activation=**tanh**)
- ❑ **Dropout** layer (**0.5** dropout rate)

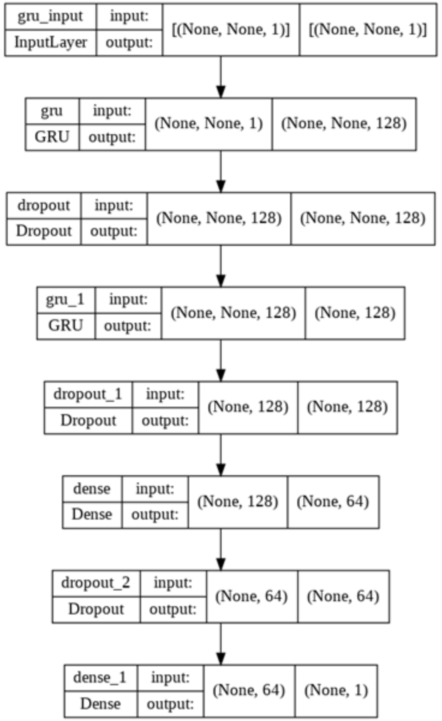
- ❑ **GRU** layer (**128** neurons, activation=**tanh**)
- ❑ **Dropout** layer (**0.5** dropout rate)

- ❑ **Dense** layer (**64** neurons)
- ❑ **Dropout** layer (**0.5** dropout rate)

- ❑ **Dense** layer (**1** neuron)

```python
# GRU model
model = Sequential()

model.add( GRU(128, input_shape=(None,1), return_sequences=True))
model.add(Dropout(0.5))

model.add( GRU(128))
model.add(Dropout(0.5))

model.add(Dense(64))
model.add(Dropout(0.5))

model.add(Dense(1))
```

CODE: GRU model



GRU model structure

Training

- All deep learning models (Deep RNN, LSTM and GRU) the training/validation//testing dataset are as follows:

  training set:

  x_train_norm, **y_train_norm**

  validation set:

  x_val_norm, **y_val_norm**

  testing set:

  x_test_norm

Training

- All deep learning models (Deep RNN, LSTM and GRU) the training parameters was set as follows:

  training parameters:

  optimizer = **adam**(learning rate=0.001, decay=1e-6)

  loss = **mean absolute error**

  (MAE was selected instead of RMSE because the error calculated from predicted output is a normalized output which is a decimal number, using RMSE will lower down an actual error, so MAE was selected to truly represent an actual error)

  performance metric = **mean absolute error**

  The best trained model was saved using **ModelCheckpoint** monitored on **minimizing validation loss**

  epochs = **100**

  batch size = **10**

Training

```python
opt = tf.keras.optimizers.Adam(lr=0.001, decay=1e-6)

# Compile model
model.compile(
    loss='mean_absolute_error',
    optimizer=opt,
    metrics=['mean_absolute_error'],
)

checkpoint_filepath = 'bestmodel.hdf5'
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint\
(filepath=checkpoint_filepath, save_weights_only=True, monitor='val_loss', mode='min', save_best_only=True)

history = model.fit(x_train_norm,
              y_train_norm,
              epochs=100,
              batch_size = 10,
              validation_data=(x_val_norm,y_val_norm),
              callbacks=[model_checkpoint_callback])
```

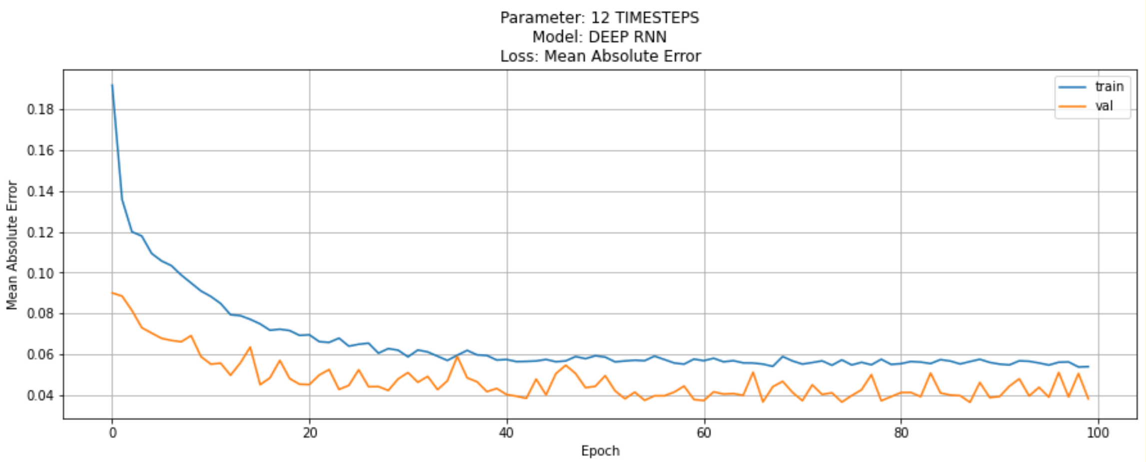
CODE: Training

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_16 (SimpleRNN)   (None, None, 128)         16640

 dropout_24 (Dropout)        (None, None, 128)         0

 simple_rnn_17 (SimpleRNN)   (None, 128)               32896

 dropout_25 (Dropout)        (None, 128)               0

 dense_16 (Dense)            (None, 64)                8256

 dropout_26 (Dropout)        (None, 64)                0

 dense_17 (Dense)            (None, 1)                 65

=================================================================
Total params: 57,857
Trainable params: 57,857
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
C:\Users\tOm\AppData\Roaming\Python\Python38\site-packages\keras\optimizer_v2\adam.py:105: UserWarning: The `lr` argument is de
precated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
144/144 [==============================] - 23s 136ms/step - loss: 0.7825 - mean_absolute_error: 0.7825 - val_loss: 0.1937 - val
_mean_absolute_error: 0.1937
Epoch 2/100
144/144 [==============================] - 16s 114ms/step - loss: 0.3380 - mean_absolute_error: 0.3380 - val_loss: 0.0793 - val
_mean_absolute_error: 0.0793
Epoch 3/100
144/144 [==============================] - 16s 110ms/step - loss: 0.2144 - mean_absolute_error: 0.2144 - val_loss: 0.0868 - val
_mean_absolute_error: 0.0868
Epoch 4/100
144/144 [==============================] - 18s 125ms/step - loss: 0.1676 - mean_absolute_error: 0.1676 - val_loss: 0.0586 - val
_mean_absolute_error: 0.0586
Epoch 5/100
144/144 [==============================] - 18s 125ms/step - loss: 0.1416 - mean_absolute_error: 0.1416 - val_loss: 0.0572 - val
_mean_absolute_error: 0.0572
Epoch 6/100
144/144 [==============================] - 16s 112ms/step - loss: 0.1209 - mean_absolute_error: 0.1209 - val_loss: 0.0617 - val
_mean_absolute_error: 0.0617
Epoch 7/100
```

Training output

Training

- For each model, the plot of MAE in training and validation set on each training epoch is plotted see how the training goes and to detect if there was any abnormal error during the training



Training set and validation set MAE on each training epoch
(Example from Deep RNN with 12 timesteps feature)

```
#plot loss, val_loss VS epoch
loss_metric = f'Mean Absolute Error'
plt.figure(figsize=(15,5))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title(f'Parameter: {feature_timestep} TIMESTEPS\n\
Model: {model_structure}\nLoss: {loss_metric}')

plt.ylabel(f'{loss_metric}')
plt.xlabel('Epoch')
plt.legend(['train','val'], loc='upper right')
plt.grid()
plt.show()
```

CODE: MAE plotting

Training

- Then the best model from training session was loaded
- The best model was tested on validation set to see how the model perform (the loss/metric from this session still represents in normalized value)

```
#lood best model
model.load_weights(checkpoint_filepath)

loss, metric = model.evaluate(x_test_norm, y_test_norm, verbose=1)
print(f'Model LOSS={loss}, METRIC={metric}')
```

```
6/6 [==============================] - 0s 23ms/step - loss: 0.0507 - mean_absolute_error: 0.0507
Model LOSS=0.050728268921375275, METRIC=0.050728268921375275
```

CODE: Loading best model and validating on validation set
(Example from Deep RNN with 12 timesteps feature)

- The actual prediction for test set was performed by the best model and then inverse transform(denormalized) to present an actual predicted temperature
- The results will be further interpreted and concluded in result section

```
y_test_predict = model.predict(x_test_norm)
y_test_predict_inv = minmax_norm.inverse_transform(y_test_predict)
```

CODE: Test set temperature prediction and denormalization

Training

- The deep learning model training time comparison (from command %%time) with machine specifications as followed are shown in the table below

- Machine Specifications:
  - CPU: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz   2.30 GHz
  - RAM: 24.0 GB (23.9 GB usable)
  - GPU: NVIDIA GeForce GTX 1050Ti

Model	Timesteps Feature		
	4	6	12
Deep RNN	**11m 28s**	**16m 6s**	**27m 54s**
LSTM	**3m 59s**	**3m 55s**	**4m 19s**
GRU	**3m 54s**	**3m 38s**	**3m 35s**

Auto Regressive model

In order to deal with the forecasting task of time-series dataset, auto regressive model is usually chosen to be the one to predict the outcome of the future values based on previous data values.

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \epsilon_t$$

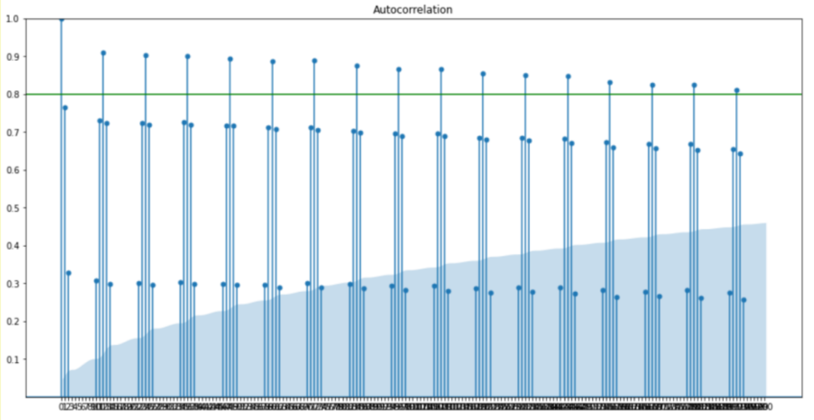
An autoregressive model uses a linear combination of past values of the target to make forecasts and is made against the target itself. Where lags express each data point across time is called a lag, Bias and weight are associated with each lag which tells the importance of that time step in predicting the final value and Autocorrelation is the relationship between forecasted variable and input variables.

Finding autocorrelation

In order to find out which lags have a strong correlation to the forecasted values, The code creates a plot that shows us how much each previous lag influences the future lag.

```
# find out which is the best lag to use
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf
import numpy as np

fig, ax = plt.subplots(figsize=(16,8))
plot_acf(df1['AverageTemperature'], lags=200, ax=ax)
plt.ylim([0,1])
plt.yticks(np.arange(0.1, 1.1, 0.1))
plt.xticks(np.arange(0, 201, 1))
plt.axhline(y=0.8, color="green")
plt.show()
```

CODE: Finding which lag is the best lag to use



Autocorrelation and lag plot

Then we provide a threshold of 0.8 to see the lag that has a very strong correlation to the future value. The lag that we are looking for is 192 which goes above the threshold.

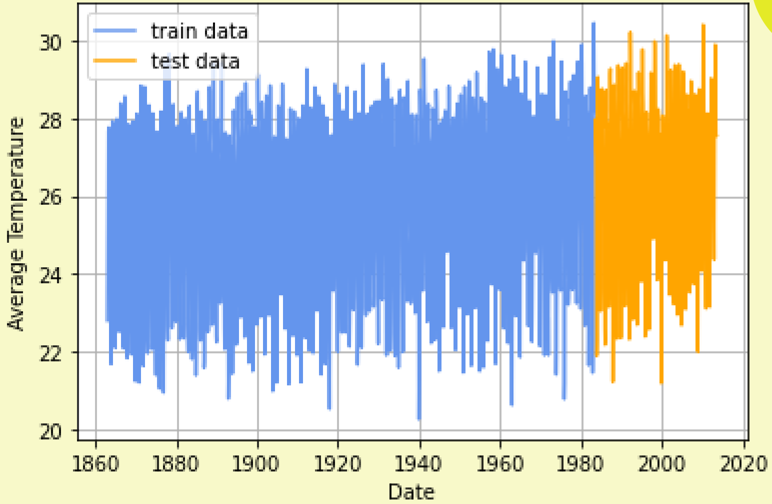
Training Auto Regressive model

```python
train_df = df1.iloc[:1446]
test_df = df1.iloc[1446:]
print('shapes:', df1.shape, train_df.shape, test_df.shape)
```

```
shapes: (1808, 1) (1446, 1) (362, 1)
```

```python
train_set_size = len(train_df)
train_set_dates = df1.head(train_set_size).index  # for plotting
test_set_dates = df1.tail(362).index

plt.grid()
plt.plot(train_set_dates, train_df.AverageTemperature, color='cornflowerblue', label='train data')
plt.plot(test_set_dates, test_df.AverageTemperature, color='orange', label='test data')
plt.legend(loc='best')
plt.xlabel('Date')
plt.ylabel('Average Temperature')
plt.show()
```

CODE: Splitting training set and testing set



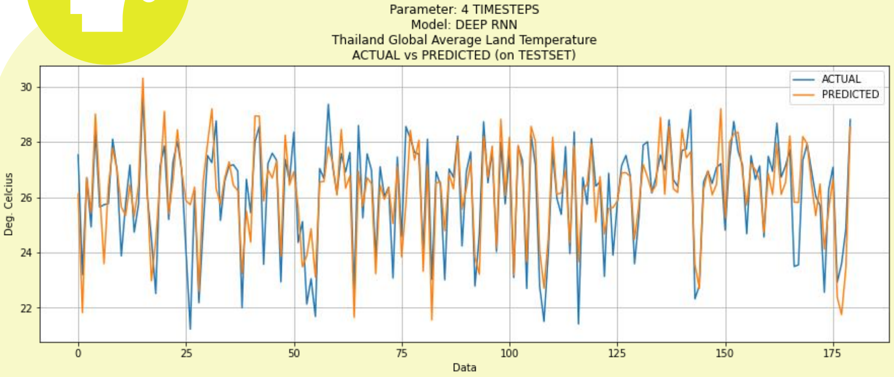
Date and time that used for training and testing

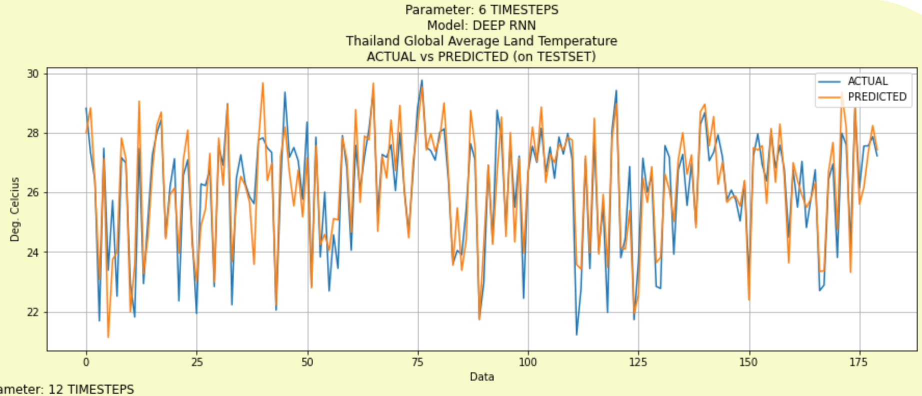
```python
from statsmodels.tsa.ar_model import AutoReg

model = AutoReg(train_df, lags=192)
trained_model = model.fit()
```

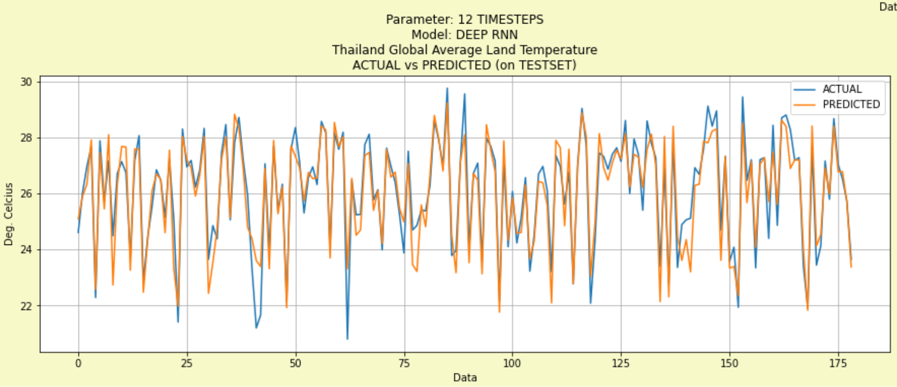
CODE: Creating and fitting the model

Results

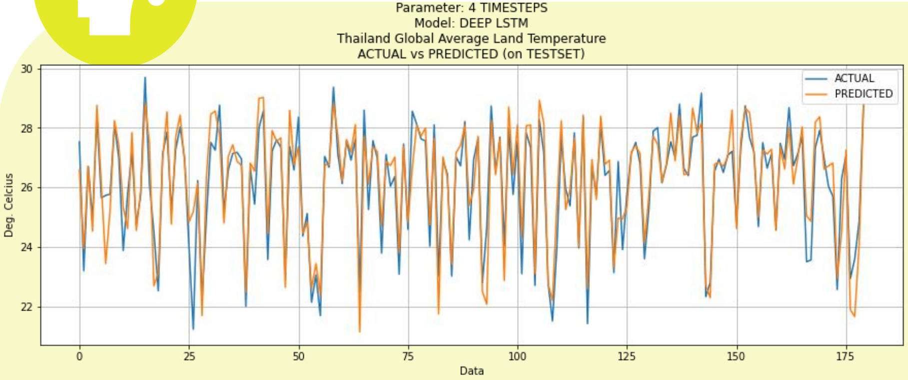
VISUALIZED RNN RESULTS FOR EACH TIMESTEP

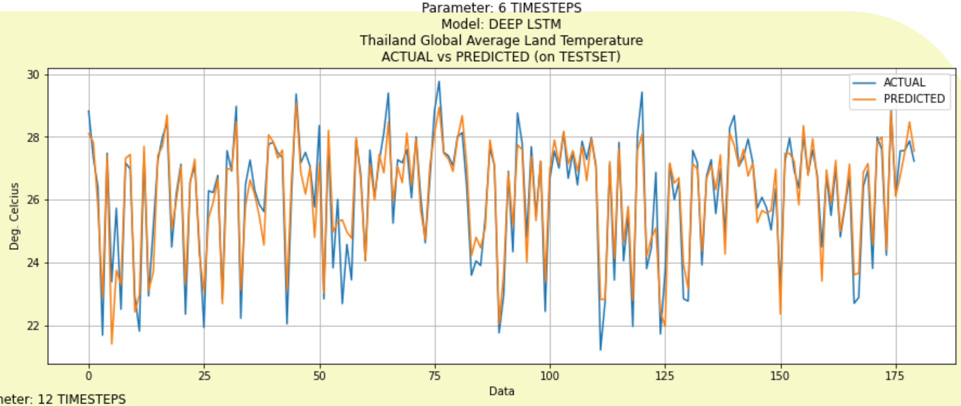


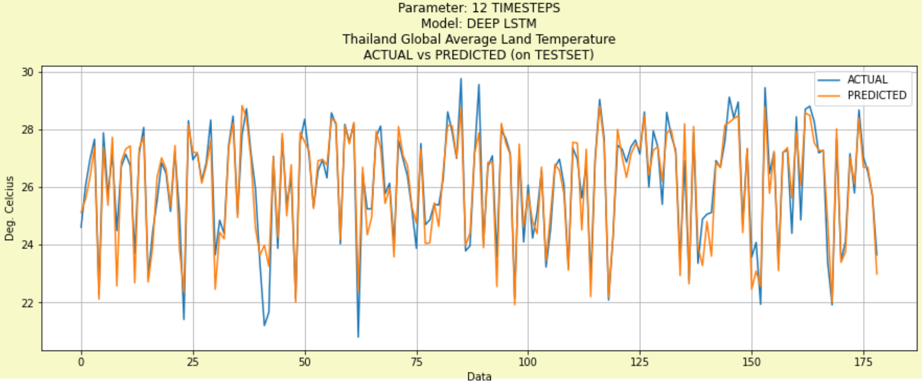




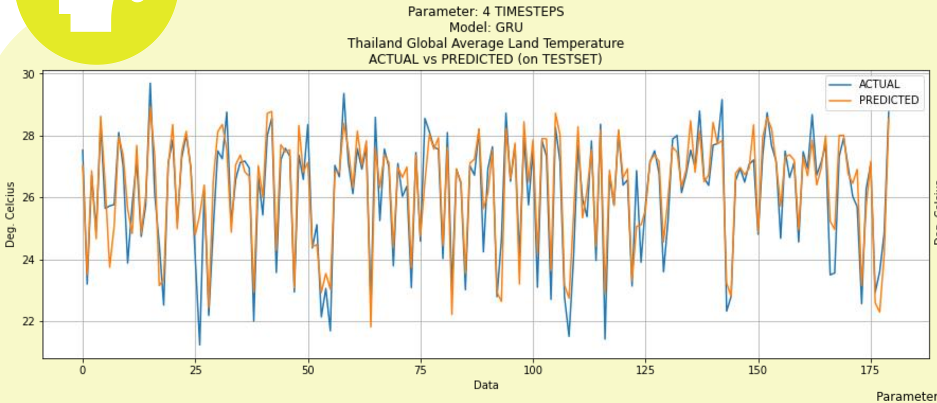
VISUALIZED LSTM RESULTS FOR EACH TIMESTEP

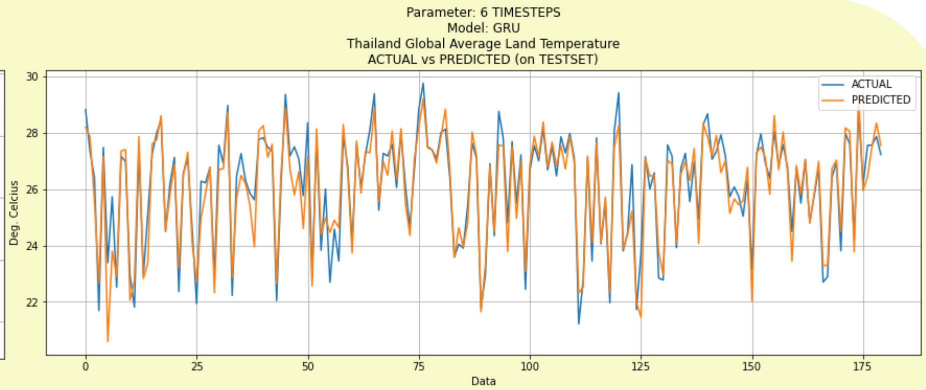


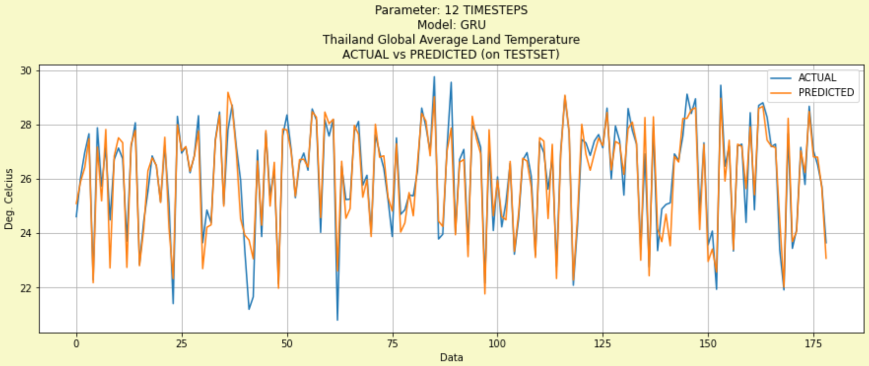




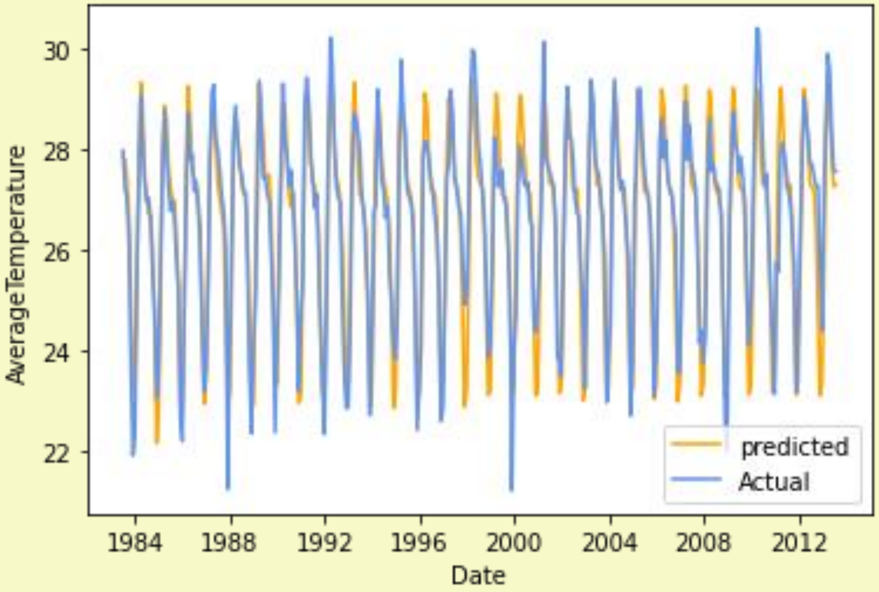
VISUALIZED GRU RESULTS FOR EACH TIMESTEP







VISUALIZED AUTOREGRESSIVE RESULTS



MAE Evaluation on Test Set

Models	4 Times	6 Times	12 Times
RNN	0.785	0.650	0.516
LTSM	0.576	0.493	0.433
GRU	0.551	0.482	0.411
AutoRegressive	0.531		

Conclusion

According to the experiment, the all deep learning's best models (Multi-layer RNN, LSTM and GRU) have better performance compared to the autoregressive method. Comparing among the deep learning model, the GRU model has minimum MAE at 0.411. The second best is LSTM at 0.433 and the last on is RNN at 0.516. We'll find that the GRU and LSTM model outperform the RNN model.

In term of input time step which is one of the important hyper-parameters, we found that the 12 time-steps input made the model the best outcome comparing to 4 and 6 time-steps input. There is a point to mentioned that the data were collected on the monthly basis and the temperature data is affected from the season of the year which may be the reason that 12 time-steps input show the best result in these experiments.

Discussion

- The results are quite good with MAE around 0.4 - 0.8 which is acceptable for temperature prediction. However, in some cases which require more accurate prediction, the configuration of GRU and LSTM model should be considered to achieve more accurate model.
- For other cases which the data may not be collected on the monthly basis, the best input time step may not be 12 as per these experiments. The input time step should be taken into account and properly fine-tuned based on each data.

References

Python Version: 3.8.5

Python Library
- Matplotlib 3.5.1
- Numpy 1.22.0
- OpenCV 4.5.5
- Pandas 1.3.0
- Tensorflow 2.7.0

Source Code
- Data preparation and model training code referenced from RNN_ex1.py (BADS7604 by Asst. Prof. Thitirat Sribiborbornratanakul)

Datasets
- From Climate Change: Earth Surface Temperature Data (https://www.kaggle.com/datasets/berkeleyearth/climate-change-earth-surface-temperature-data?select=GlobalTemperatures.csv) with license: Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)

Citing

For those who want to use any presentation images please reference the image in **bibtex** format

Team Members

6310412018	Pongsarat Chootai	**(25%)** Highlights, Data preparation, Network Architecture, Experimental variation on LSTM, Conclusion and Discussion
6310412021	Theethut Narksenee	**(25%)** Data exploration, Autoregressive model, and Report (Collecting and comparing the results)
6310412024	Saranchai Angkawinijwong	**(25%)** **Programming**: Develop Data Preparation and Deep Learning Base Code **Presentation**: Data Preparation, Model Architecture(Deep RNN), Training(Deep Learning) and Miscellaneous(References, Citing, End Credit)
6420400001	Krittipat Chuenphitthayavut	**(25%)** Introduction and Highlights composition, Data preparation, Network Architecture, Experimental variation on GRU mode

End Credit

This project is a part of **Deep Learning** course

Course code: **BADS7604**

Syllabus: Business Analytics and Data Science (**BADS**)

Institute: National Institute of Development Administration (**NIDA**)

Semester: **2/2021**

Thank you for your attention