

Vehicle CO₂ Emission Prediction

Deep Learning
Assignment 1

Team Members

6310412018

Pongsarat
Chootai

(25%)

Find dataset, Data cleansing and preparation, Develop Linear Regression Model, Develop initial Deep Learning Model, Result Summarization

6310412021

Theethut
Narksenee

(25%)

Experimental variation on activation fn. and learning rate

6310412024

Saranchai
Angkawinijwong

(25%)

Experimental variation on number of neuron and number of layer

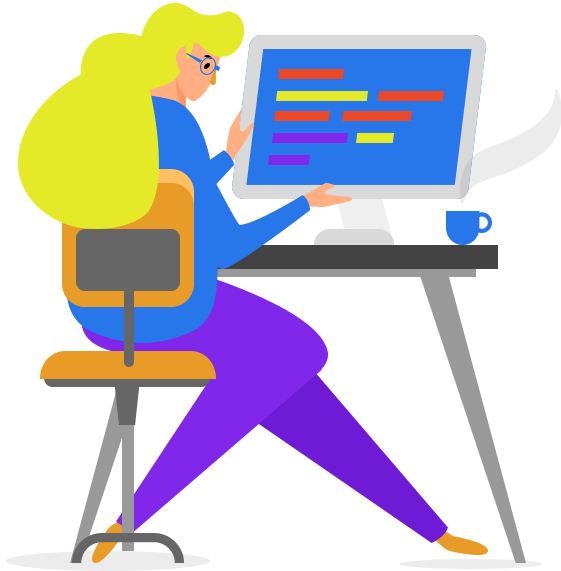
6420400001

Krittirat
Chuenphitthayavut

(25%)

Experimental variation on number of epoch and batch size

Agenda



01

Team Members

02

Why CO₂ Emission ?

03

**Data Exploration
and Preparation**

04

Linear Regression

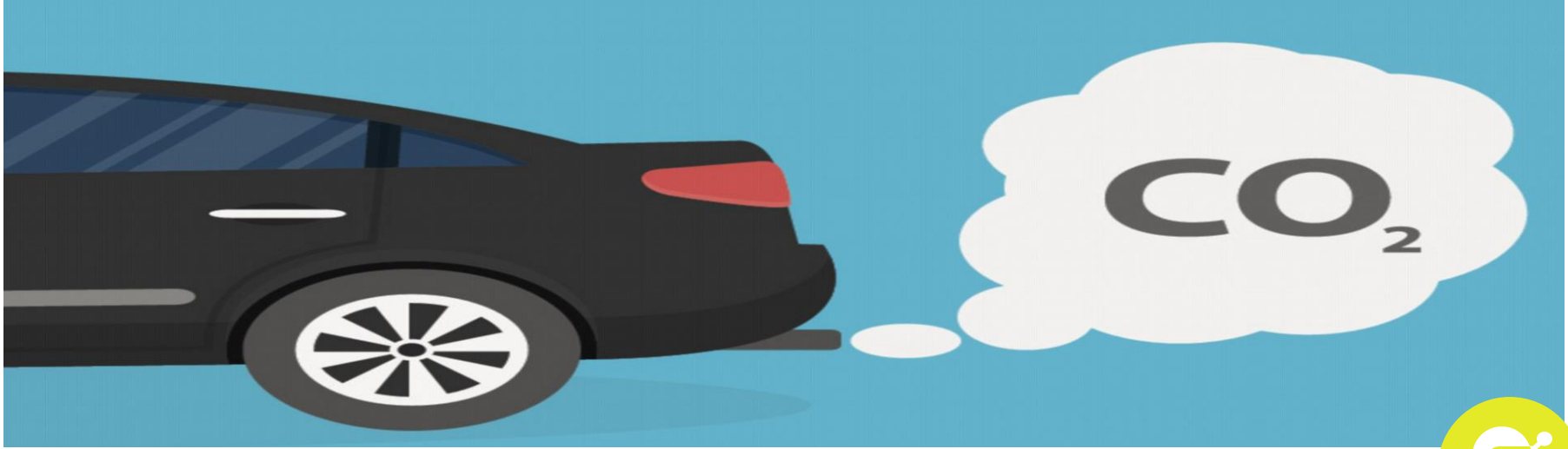
05

Deep Learning

06

Summary

Why CO₂ Emission ?



The CO₂ emission is the most concerned environmental problem for our earth for a long time. According to UN Climate Change Conference (COP26), all countries are going to achieve “Net Zero” which means the overall CO₂ emission equals to 0 with in 2040 – 2070 to ensure that the atmosphere won’t be damaged until it’ll be irreversible anymore.

The CO₂ emission from vehicles is one of the main CO₂ emission source which will be benefit if we can estimate them accurately in order to plan for CO₂ net-zero in near future.



Dataset



The data source is a public data from Canada government as per below link
<https://www.kaggle.com/debajyotipodder/co2-emission-by-vehicles> which comprises of 7385 data of vehicles with 11 features each.

Import data

```
1 #import csv file
2 df = pd.read_csv('CO2 Emissions_Canada.csv')
3 df.tail(5)
```

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)	Fuel Consumption Comb (L/100 km)	Fuel Consumption Comb (mpg)	CO2 Emissions(g/km)
7380	VOLVO	XC40 T5 AWD	SUV - SMALL	2.0	4	AS8	Z	10.7	7.7	9.4	30	219
7381	VOLVO	XC60 T5 AWD	SUV - SMALL	2.0	4	AS8	Z	11.2	8.3	9.9	29	232
7382	VOLVO	XC60 T6 AWD	SUV - SMALL	2.0	4	AS8	Z	11.7	8.6	10.3	27	240
7383	VOLVO	XC90 T5 AWD	SUV - STANDARD	2.0	4	AS8	Z	11.2	8.3	9.9	29	232
7384	VOLVO	XC90 T6 AWD	SUV - STANDARD	2.0	4	AS8	Z	12.2	8.7	10.7	26	248

Data Exploration and Preparation



According to the data, there are some categorical data as shown below which will be converted as dummy variables.

```
1 #create dummy variables for categorical variables
2 df_final = pd.get_dummies(df)
3 df_final.tail()
4
```

Vehicle class

STATION WAGON - SMALL
COMPACT
MID-SIZE
SUV - SMALL
MINICOMPACT
SPECIAL PURPOSE VEHICLE
STATION WAGON - MID-SIZE
SUBCOMPACT
MINIVAN
FULL-SIZE
TWO-SEATER
PICKUP TRUCK - SMALL
PICKUP TRUCK - STANDARD
SUV - STANDARD
VAN - CARGO
VAN - PASSENGER

Fuel type

X = regular gasoline
Z = premium gasoline
D = diesel
E = ethanol (E85)
N = natural gas

Vehicle Class_COMPACT	Vehicle Class_FULL- SIZE	Vehicle Class_MID- SIZE	Vehicle Class_MINICOMPACT	Vehicle Class_MINIVAN	Vehicle Class_PICKUP TRUCK - SMALL	Vehicle Class_PICKUP TRUCK - STANDARD	Vehicle Class_SPECIAL PURPOSE VEHICLE	Vehicle Class_STATION WAGON - MID- SIZE	Vehicle Class_STATION WAGON - SMALL	Vehicle Class_SUBCOMPACT	Vehicle Class_SUV - SMALL	Vehicle Class_SUV - STANDARD	Vehicle Class_TWO- SEATER	Vehicle Class_VAN - CARGO	Vehicle Class_VAN - PASSENGER	Fuel Type_D	Fuel Type_E	Fuel Type_N	Fuel Type_X	Fuel Type_Z
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

Data Exploration and Preparation



Next is to normalize the input. We select min – max scalar method to convert the input variables' value to be in range 0 – 1 since all those values shall be in positive number only.

```
1 #normalization
2 cols_to_norm = ['Engine Size(L)','Cylinders','Fuel Consumption City (L/100 km)','Fuel Consumption Hwy (L/100 km)','Fuel Consumption Comb (L/100 km)','Fuel Consumption Comb (mpg)']
3 df_final[cols_to_norm] = MinMaxScaler().fit_transform(df_final[cols_to_norm])
4
5 #relocate output column
6 df_final['CO2 Emission(g/km)'] = df_final['CO2 Emissions(g/km)']
7 df_final = df_final.drop(columns=['CO2 Emissions(g/km)'])
8
9 df_final.tail(5)
```

	Engine Size(L)	Cylinders	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)	Fuel Consumption Comb (L/100 km)	Fuel Consumption Comb (mpg)	Vehicle Class_COMPACT	Vehicle Class_FULL-SIZE	Vehicle Class_MID-SIZE	Vehicle Class_MINICOMPACT	Vehicle Class_MINIVAN	Vehicle Class_PICKUP TRUCK - SMALL	Vehicle Class_PICKUP TRUCK - STANDARD	Vehicle Class_SPECIAL PURPOSE VEHICLE
7380	0.146667	0.076923	0.246212	0.222892	0.240909	0.327586	0	0	0	0	0	0	0	0
7381	0.146667	0.076923	0.265152	0.259036	0.263636	0.310345	0	0	0	0	0	0	0	0
7382	0.146667	0.076923	0.284091	0.277108	0.281818	0.275862	0	0	0	0	0	0	0	0
7383	0.146667	0.076923	0.265152	0.259036	0.263636	0.310345	0	0	0	0	0	0	0	0
7384	0.146667	0.076923	0.303030	0.283133	0.300000	0.258621	0	0	0	0	0	0	0	0

We performed train – test split with 80% training data and 20% test data. The variable x_train, y_train, x_test, and y_test are ready to be analyzed with linear regression as based line and deep learning onward.

```
[10] 1 #train test split
      2 test_size = 0.2
      3 train, test = train_test_split(df_final, test_size = test_size, random_state = 3)
```

```
1 #define x and y
2 x_train = train.iloc[:, 0:27]
3 y_train = train.iloc[:, 27]
4 x_test = test.iloc[:, 0:27]
5 y_test = test.iloc[:, 27]
```

Linear Regression



After fine-tuned linear regression model, the result shows the RMSE = 4.5507 with $R^2 = 0.9938$

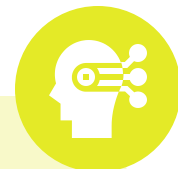
Linear Regression

```
[15] 1 #Import libraries
      2 from sklearn.linear_model import LinearRegression
      3 from sklearn.metrics import mean_squared_error

[17] 1 #Create model
      2 model_linearreg = LinearRegression().fit(x_train, y_train)
      3
      4 #Predict y
      5 y_predict = model_linearreg.predict(x_test)
      6
      7 #RMSE
      8 print('RMSE:', round(mean_squared_error(y_test, y_predict, squared=False),4))
      9
     10 #R^2
     11 print('R-Squared:', round(r2_score(y_test,y_predict),4))

RMSE: 4.5507
R-Squared: 0.9938
```


Deep Learning

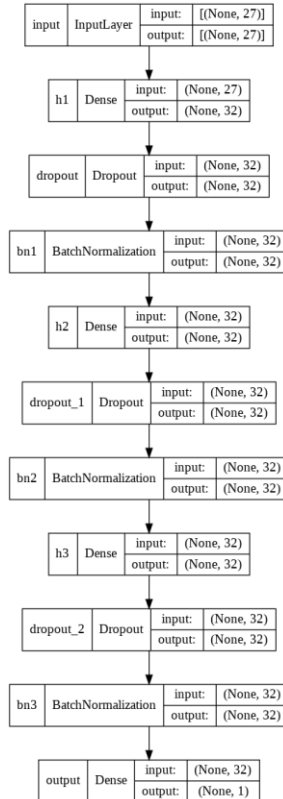


We construct the MLP deep learning model with initial parameters:

- neuron = 32
- layer = 3
- batch_size = 50
- epoch = 200
- activation = 'relu'
- learning_rate = 0.001
- momentum_decay = 0.9

```
1 #DL BASELINE
2 neuron = 32
3 layer = 3
4 batch_size = 50
5 epoch = 200
6 activation = 'relu'
7 learning_rate = 0.001
8 momentum_decay = 0.9 #default : https://keras.io/api/optimizers/adam/
9 model_name = 'baseline_dl'
10
11 #BUILD MODEL
12 keras.backend.clear_session()
13
14 model = tf.keras.models.Sequential()
15
16 #####
17 model.add(tf.keras.Input(shape=(x_train.shape[1]), name='input'))
18
19 name_layer = 0
20 for i in range(layer):
21     name_layer += 1
22     model.add(tf.keras.layers.Dense(neuron, activation = activation, kernel_initializer = 'glorot_uniform', name = f'h{name_layer}'))
23     model.add(tf.keras.layers.Dropout(0.5))
24     model.add(tf.keras.layers.BatchNormalization(axis = -1, name = f'bn{name_layer}'))
25
26 model.add(tf.keras.layers.Dense(1, name='output'))
27 #####
28
29 model.summary()
30
31 model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate, beta_1=momentum_decay, beta_2=0.999, epsilon=1e-07, amsgrad=False, name='Adam'), \
32               loss='mean_squared_error', metrics=[tf.keras.metrics.RootMeanSquaredError()])
33
34 # Model weights are saved at the end of every epoch, if it's the best seen so far.
35 #checkpoint_filepath = '/gdrive/MyDrive/Colab Notebooks/BAD57604/model/bestmodel_epoch(epoch:82d)_valloss{val_loss:2f}.hdf5'
36 if not use_gdrive:
37     #local drive
38     checkpoint_filepath = f'bestmodel_{model_name}.hdf5'
39 else:
40     #google drive
41     checkpoint_filepath = f'/gdrive/MyDrive/Colab Notebooks/BAD57604/model/bestmodel_{model_name}.hdf5'
42
43 model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_filepath, save_weights_only=True, monitor='loss', mode='min', save_best_only=True)
44 model.fit(x_train, y_train, batch_size=batch_size, epochs=epoch, verbose=1, validation_split=0.2, callbacks=[model_checkpoint_callback])
45
46 #Load best saved model
47 #https://keras.io/api/callbacks/model\_checkpoint/
48 model.load_weights(checkpoint_filepath)
49
50 loss, metric = model.evaluate(x_test, y_test, verbose=1)
51 print(f'Model({neuron} neuron, {layer} layer): LOSS={loss}, METRIC={metric}')
52
53 keras.backend.clear_session()
54
```

Deep Learning



Model: "sequential"

Layer (type)	Output Shape	Param #
h1 (Dense)	(None, 32)	896
dropout (Dropout)	(None, 32)	0
bn1 (BatchNormalization)	(None, 32)	128
h2 (Dense)	(None, 32)	1056
dropout_1 (Dropout)	(None, 32)	0
bn2 (BatchNormalization)	(None, 32)	128
h3 (Dense)	(None, 32)	1056
dropout_2 (Dropout)	(None, 32)	0
bn3 (BatchNormalization)	(None, 32)	128
output (Dense)	(None, 1)	33

=====
Total params: 3,425
Trainable params: 3,233
Non-trainable params: 192

Epoch 200/200
95/95 [=====] - 1s 8ms/step - loss: 570.8145 - root_mean_squared_error: 23.8917 - val_loss: 187.9370 - val_root_mean_squared_error: 13.7890
47/47 [=====] - 0s 4ms/step - loss: 171.5871 - root_mean_squared_error: 13.0991
Model(32 neuron, 3 layer): LOSS=171.58705139160156, METRIC=13.099124908447266



The model architecture is shown here with initial RMSE = 13.10

Neurons and Layers Variation



Variation Specifications

- Neuron's variation = [32, 128, 512, 2048]
- Layers variation = [3, 7, 11, 15]
- Each layer consists of
 - 1) one Dense Layer
 - 2) one Dropout(prob=0.5)
 - 3) one Batch Normalization(axis=-1)

MODEL for 3 LAYERS, 32 NEURONS
Model: "sequential"

Layer (type)	Output Shape	Param #
h1 (Dense)	(None, 32)	896
dropout (Dropout)	(None, 32)	0
bn1 (BatchNormalization)	(None, 32)	128
h2 (Dense)	(None, 32)	1056
dropout_1 (Dropout)	(None, 32)	0
bn2 (BatchNormalization)	(None, 32)	128
h3 (Dense)	(None, 32)	1056
dropout_2 (Dropout)	(None, 32)	0
bn3 (BatchNormalization)	(None, 32)	128
output (Dense)	(None, 1)	33
=====		
Total params: 3,425		
Trainable params: 3,233		
Non-trainable params: 192		

Example:

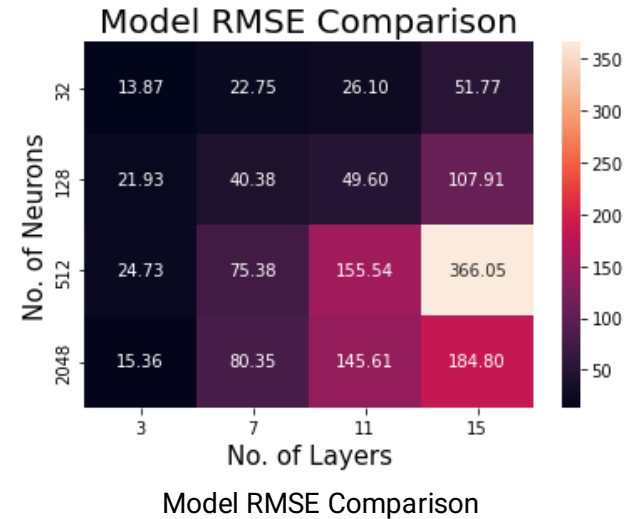
Model summary of 32 neurons and 3 layers

Neurons and Layers Variation



Results

- Model with 32 neurons and 3 layers gives the best fit with 13.87 root mean square error
- Adding more neuron or more layer tend to worsen the model efficiency
- This can be caused from the data overfitting



Neurons and Layers Variation



Code

```
#VARY NEURONS AND LAYERS BY SARANCHAI (TOM)
#DL BASELINE
#neuron = 32
#layer = 3
batch_size = 50
epoch = 200
activation = 'relu'
learning_rate = 0.001
momentum_decay = 0.9 #default: https://keras.io/api/optimizers/adam/
#model_name = 'baseline_dl'
num_of_neuron = [32, 128, 512, 2048]
num_of_layer = [3, 7, 11, 15]

keras.backend.clear_session()

loss_list_all = []
metric_list_all = []

for neuron in num_of_neuron:
    loss_list = []
    metric_list = []
    for layer in num_of_layer:
        print(f'MODEL for {layer} LAYERS, {neuron} NEURONS')
        model = tf.keras.models.Sequential()

        #####
        model.add(tf.keras.Input(shape=(x_train.shape[1]), name='input'))

        name_layer = 0
        for i in range(layer):
            name_layer += 1
            model.add(tf.keras.layers.Dense(neuron, activation = activation, kernel_initializer = 'glorot_uniform', name = f'h{name_layer}'))
            model.add(tf.keras.layers.Dropout(0.5))
            model.add(tf.keras.layers.BatchNormalization(axis = -1, name = f'bn{name_layer}'))

        #model.add(tf.keras.layers.Dropout(0.5))

        model.add(tf.keras.layers.Dense(1, name='output'))
        #####

        model.summary()

    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate, beta_1=momentum_decay, beta_2=0.999, epsilon=1e-08),
                  loss='mean_squared_error', metrics=[tf.keras.metrics.RootMeanSquaredError()])
```

```
# Model weights are saved at the end of every epoch, if it's the best seen so far.
#checkpoint_filepath = '/gdrive/MyDrive/Colab Notebooks/BADS7604/model/bestmodel_epoch{epoch:02d}_val_loss{val_loss:2f}.hdf5'
if not use_gdrive:
    #local drive
    checkpoint_filepath = f'model/bestmodel_{layer}layers_{neuron}neurons.hdf5'
else:
    #google drive
    checkpoint_filepath = f'/gdrive/MyDrive/Colab Notebooks/BADS7604/model/bestmodel_{layer}layers_{neuron}neurons.hdf5'

model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_filepath, save_weights_only=True, monitor=
model.fit(x_train, y_train, batch_size=batch_size, epochs=epoch, verbose=1, validation_split=0.2, callbacks=[model_checkpoint

#Load best model
#https://keras.io/api/callbacks/model_checkpoint/
model.load_weights(checkpoint_filepath)

loss, rmse = model.evaluate(x_test, y_test, verbose=True)
print(f'Model({neuron} neuron, {layer} layer): LOSS={loss}, RMSE={rmse}')

loss_list.append(loss)
metric_list.append(rmse)

keras.backend.clear_session()
loss_list_all.append(loss_list)
metric_list_all.append(metric_list)

print(loss_list_all)
print(metric_list_all)
```

Activation function and Learning rate variation



Variation Specifications

- Activation function's variation = [Softmax, Softplus, Softsign, ReLu, Tanh, sigmoid, linear]
- The scikit-learn library has a unified model scoring system where it assumes that all model scores are maximized. In order this system to work with scores that are minimized, like MSE, the scores that are minimized are inverted by making them negative. Thus, we see the negative result of MSE

```
val_root_mean_squared_error: 7.6258
Best: -48.181338 using {'activation': 'linear'}
-64.699009 (13.444998) with: {'activation': 'softmax'}
-1058.227030 (72.708602) with: {'activation': 'softplus'}
-116.058400 (11.461833) with: {'activation': 'softsign'}
-211.344910 (31.994278) with: {'activation': 'relu'}
-75.060771 (12.338075) with: {'activation': 'tanh'}
-326.523326 (51.857492) with: {'activation': 'sigmoid'}
-48.181338 (7.597578) with: {'activation': 'linear'}
```

Result of activation function Gridsearching

```
def create_model(activation='relu'):
    """ create model """
    keras.backend.clear_session()

    model = tf.keras.models.Sequential()

    model.add(tf.keras.layers.Input(shape=(x_train.shape[1],)))

    model.add(tf.keras.layers.Dense(32, activation=activation, kernel_initializer='glorot_uniform', name='h1'))
    model.add(tf.keras.layers.Dropout(0.5))
    model.add(tf.keras.layers.BatchNormalization(axis=-1, name='bn1'))
    model.add(tf.keras.layers.Dense(32, activation=activation, kernel_initializer='glorot_uniform', name='h2'))
    model.add(tf.keras.layers.Dropout(0.5))
    model.add(tf.keras.layers.BatchNormalization(axis=-1, name='bn2'))
    model.add(tf.keras.layers.Dense(32, activation=activation, kernel_initializer='glorot_uniform', name='h3'))
    model.add(tf.keras.layers.Dropout(0.5))
    model.add(tf.keras.layers.BatchNormalization(axis=-1, name='bn3'))

    model.add(tf.keras.layers.Dense(1, name='output'))

    """ Compile model """
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=0.00000001),
                  loss='mean_squared_error',
                  metrics=[tf.keras.metrics.RootMeanSquaredError()])

    return model

seed = 1
np.random.seed(seed)
tf.random.set_seed(seed)

model = KerasRegressor(build_fn=create_model, batch_size=50, epochs=200, verbose=1, validation_split=0.2)
activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'linear']
param_grid = dict(activation=activation)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(x_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %s" % (mean, stdev, param))
```

Code for tuning activation function parameter

Activation function and Learning rate variation



Variation Specifications

- Learning rate = [0.0001 0.001 0.01 0.1]
- Setting default learning rate to 0.001 and all hyperparameter according to baseline using Adam optimizer

```
val_root_mean_squared_error: 14.0423
Best: -196.926361 using {'learn_rate': 0.001}
-30849.298177 (177.441396) with: {'learn_rate': 0.0001}
-196.926361 (7.914879) with: {'learn_rate': 0.001}
-137840.940020 (194673.506420) with: {'learn_rate': 0.01}
-1303206.953959 (1829770.584877) with: {'learn_rate': 0.1}
```

Result of learning rate Gridsearching

```
def create_model(learn_rate=0.001):
    """ # create model
    keras.backend.clear_session()

    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Input(shape=(x_train.shape[1],)))

    model.add(tf.keras.layers.Dense(32, activation='relu', kernel_initializer='glorot_uniform', name='h1'))
    model.add(tf.keras.layers.Dropout(0.5))
    model.add(tf.keras.layers.BatchNormalization(axis=-1, name='bn1'))
    model.add(tf.keras.layers.Dense(32, activation='relu', kernel_initializer='glorot_uniform', name='h2'))
    model.add(tf.keras.layers.Dropout(0.5))
    model.add(tf.keras.layers.BatchNormalization(axis=-1, name='bn2'))
    model.add(tf.keras.layers.Dense(32, activation='relu', kernel_initializer='glorot_uniform', name='h3'))
    model.add(tf.keras.layers.Dropout(0.5))
    model.add(tf.keras.layers.BatchNormalization(axis=-1, name='bn3'))

    model.add(tf.keras.layers.Dense(1, name='output'))

    """ # Compile model
    # optimizer = SGD(lr=learn_rate, momentum=momentum)
    optimizer = tf.keras.optimizers.Adam(lr=learn_rate)
    model.compile(optimizer=optimizer,
                  loss='mean_squared_error',
                  metrics=[tf.keras.metrics.RootMeanSquaredError()])

    return model

seed = 1
np.random.seed(seed)
tf.random.set_seed(seed)

model = KerasRegressor(build_fn=create_model, batch_size=50, epochs=200, verbose=1, validation_split=0.2)
learn_rate = [0.0001, 0.001, 0.01, 0.1]
# momentum = [0.0, 0.2, 0.4, 0.6]
# param_grid = dict(learn_rate=learn_rate, momentum=momentum)
param_grid = dict(learn_rate=learn_rate)

grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(x_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %s" % (mean, stdev, param))
```

Code for tuning learning rate parameter

Deep Learning hyperparameter tuning

Neurons variation = [**32**, 128, 512, 2048]

Layers variation = [**3**, 7, 11, 15]

Batch_size = **50**, 100, 150, **200**

Epoch = **100**, 150, **200**

Activation : Softmax, Softplus, Softsign, **ReLu**, Tanh, sigmoid, **linear**

Learning rate : 0.0001 **0.001** 0.01 0.1

Verbose = 1

Optimizer = **Adam**

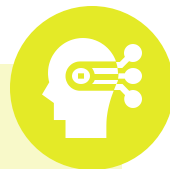
Loss function = **MSE**

Metric = **RMSE**

Save_best_only = True

**Noted : default in bold
and selected parameter in Red**

Summary



According to the result, there is an interesting point needed discussion. You'll find that the result from linear regression has the $RMSE = 4.55$ comparing to the best result from deep learning $RMSE = 5.11$. It's interesting that the result from linear regression has lower RMSE value. We've discussed and found that the prediction of CO_2 from various variables may be not complex and be in the form of some correlation. Therefore, the linear regression model perform well this time.

What we learned from this experiments is how we change the deep learning parameters and monitor the performance of the model. Finally, we learned that the deep learning model won't outperform other techniques every time. We just have to pick the most suited model for each situations.