modelNotebook

May 28, 2019

In [1]:
```
#!/usr/bin/env python3
import numpy as np
import math
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

from keras.utils import to_categorical
from keras.utils import plot_model
import IPython.display
import matplotlib.pyplot as plt

import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
Using TensorFlow backend.
```

In [2]:
```
#if i want to speed up and use one file
#files2 = "CSVLog_20190518_122417.csv"

#i aggregate the data points for five files into aggregate.csv. i have removed the tim
files = "aggregate_no_time_column.csv"

raw_dataset = pd.read_csv(files, sep=",", skipinitialspace=True)
dataset = raw_dataset.copy()

#remove whitespace in front of column name
dataset.columns = dataset.columns.str.lstrip()
```

In [3]:
```
dataset.tail()
```

Out[3]:

	Absolute load value (%)	Absolute throttle position (%)	\
6633	18.03922	11.76471	
6634	17.64706	11.76471	
6635	17.64706	11.37255	

6636	17.25490	11.37255
6637	17.64706	11.37255

	Ambient air temperature (F)	Barometric pressure (inHg)	\
6633	59.0	29.8254	
6634	59.0	29.8254	
6635	59.0	29.8254	
6636	59.0	29.8254	
6637	59.0	29.8254	

	Calculated load value (%)	Commanded fuel rail pressure A (inHg)	\
6633	29.01961	885.903	
6634	29.01961	885.903	
6635	29.41176	885.903	
6636	29.41176	885.903	
6637	29.80392	885.903	

	Engine coolant temperature (F)	Engine RPM (RPM)	Fuel level input (%)	\
6633	183.2	658.75	67.05882	
6634	183.2	640.00	67.05882	
6635	183.2	607.75	67.05882	
6636	183.2	606.75	67.05882	
6637	183.2	604.50	67.05882	

	Fuel rail pressure (psi)	Fuel/Air commanded equivalence ratio	\
6633	424.9606	0.994239	
6634	449.6170	0.994239	
6635	429.3117	0.994239	
6636	449.6170	0.994239	
6637	429.3117	0.994239	

	Ignition timing advance for #1 cylinder (deg)	\
6633	5.0	
6634	4.5	
6635	4.0	
6636	6.5	
6637	6.0	

	Intake air temperature (F)	Intake manifold absolute pressure (inHg)	\
6633	71.6	11.51674	
6634	71.6	11.51674	
6635	71.6	11.51674	
6636	71.6	11.51674	
6637	71.6	11.51674	

	Long term fuel % trim - Bank 1 (%)	Mass air flow rate (lb/min)	\
6633	-0.78125	0.308259	
6634	-0.78125	0.257985	

```
6635                            -0.78125                    0.325458
6636                            -0.78125                    0.239463
6637                            -0.78125                    0.227556

      Vehicle speed (MPH)  Fuel rate (gal/hr)  Instant fuel economy (MPG)  \
6633                  0.0            0.205909                         0.0
6634                  0.0            0.172327                         0.0
6635                  0.0            0.217398                         0.0
6636                  0.0            0.159955                         0.0
6637                  0.0            0.152002                         0.0

      Total fuel economy (MPG)
6633                  30.11613
6634                  30.11534
6635                  30.11455
6636                  30.11384
6637                  30.11316
```

In [4]:
```
%%time
#set seed here!

#creating data and splitting and random shuffling
#https://stackoverflow.com/questions/38250710/how-to-split-data-into-3-sets-train-vali
def train_validate_test_split(df, train_percent=.6, validate_percent=.2, seed=None):
    np.random.seed(seed)
    perm = np.random.permutation(df.index)
    m = len(df.index)
    train_end = int(train_percent * m)
    validate_end = int(validate_percent * m) + train_end
    train = df.loc[perm[:train_end]]
    validate = df.loc[perm[train_end:validate_end]]
    test = df.loc[perm[validate_end:]]
    return train, validate, test

np.random.seed(88)
train, validate, test = train_validate_test_split(dataset)

train_labels = train.pop("Total fuel economy (MPG)")
validate_labels = validate.pop("Total fuel economy (MPG)")
test_labels = test.pop("Total fuel economy (MPG)")
```

```
CPU times: user 4.45 ms, sys: 4.95 ms, total: 9.4 ms
Wall time: 8.03 ms
```

In [5]:
```
#test_labels
#train.dtypes
```

In [6]:
```
train = train.apply(lambda col:pd.to_numeric(col, errors='coerce'))
train_stats = train.describe(include = 'all')
```

```
#train_stats = train.transpose()
train_stats
```

Out[6]:

	Absolute load value (%)	Absolute throttle position (%)
count	3982.000000	3982.000000
mean	34.765365	21.341330
std	24.278678	13.898859
min	0.000000	0.000000
25%	16.078430	12.156860
50%	24.705880	15.294120
75%	49.411770	26.666670
max	99.215680	90.980390

	Ambient air temperature (F)	Barometric pressure (inHg)
count	3982.000000	3982.000000
mean	67.695429	29.726472
std	12.542581	0.509203
min	0.000000	0.000000
25%	62.600000	29.530100
50%	62.600000	29.825400
75%	66.200000	29.825400
max	118.400000	30.120700

	Calculated load value (%)	Commanded fuel rail pressure A (inHg)
count	3982.000000	3982.000000
mean	42.617465	1844.860021
std	26.284888	1166.247340
min	0.000000	0.000000
25%	20.000000	885.903000
50%	38.823530	1213.687000
75%	56.470590	2571.333750
max	100.000000	5906.020000

	Engine coolant temperature (F)	Engine RPM (RPM)	Fuel level input (%)
count	3982.000000	3982.000000	3982.000000
mean	185.631592	1512.254332	74.660482
std	7.621659	643.830795	10.860770
min	0.000000	0.000000	0.000000
25%	183.200000	1075.937500	67.450980
50%	185.000000	1480.250000	74.509800
75%	188.600000	1982.937500	81.176470
max	204.800000	4719.500000	93.333340

	Fuel rail pressure (psi)	Fuel/Air commanded equivalence ratio
count	3982.000000	3982.000000
mean	922.304889	0.981457
std	568.094712	0.031924
min	0.000000	0.893070

```
25%                  478.624500                                0.990426
50%                  617.860800                                0.993202
75%                 1261.828000                                0.995550
max                 2981.976000                                1.016260

       Ignition timing advance for #1 cylinder (deg)  \
count                                    3982.000000
mean                                       21.181065
std                                        15.263267
min                                       -10.000000
25%                                         4.500000
50%                                        24.500000
75%                                        35.500000
max                                        47.500000

       Intake air temperature (F)  Intake manifold absolute pressure (inHg)  \
count                 3982.000000                                3982.000000
mean                    73.401457                                  17.569520
std                     13.998863                                   8.846116
min                      0.000000                                   0.000000
25%                     64.400000                                   9.744933
50%                     68.000000                                  15.060350
75%                     77.000000                                  27.167690
max                    122.000000                                  30.416000

       Long term fuel % trim - Bank 1 (%)  Mass air flow rate (lb/min)  \
count                         3982.000000                  3982.000000
mean                             5.335337                     1.484214
std                              3.947341                     1.565152
min                             -2.343750                     0.050274
25%                              2.343750                     0.312559
50%                              5.468750                     0.792477
75%                              7.812500                     2.217348
max                             12.500000                    11.959920

       Vehicle speed (MPH)  Fuel rate (gal/hr)  Instant fuel economy (MPG)
count          3982.000000         3982.000000                 3982.000000
mean             31.360988            1.002811                   61.641141
std              22.609982            1.053180                   79.944052
min               0.000000            0.033585                    0.000000
25%              11.184680            0.209994                   11.185377
50%              31.068560            0.542141                   32.477560
75%              45.981470            1.494769                   85.577900
max              76.428660            8.026347                 1128.966000
```

In [7]:
```
type(train_stats)
```

Out[7]:
```
pandas.core.frame.DataFrame
```

In [8]:
```
#https://stackoverflow.com/questions/40758562/can-anyone-explain-me-standardscaler
scaler = StandardScaler()
normed_train = scaler.fit_transform(train)
normed_validate = scaler.fit_transform(validate)
normed_test = scaler.fit_transform(test)

#not technically correct
#from tf regression website
#Note: Although we intentionally generate these statistics from only the training data
#these statistics will also be used to normalize the test dataset. We need to do that
#dataset into the same distribution that the model has been trained on.

#I normalize it according to the mean and sd of each predictor from each set instead o
#training set only
```

In [9]:
```
#debugging
#normed_train
```

In [10]:
```
def build_model_A():
  model = keras.Sequential([
    layers.Dense(5, activation=tf.nn.relu, input_shape=[len(train.keys())]),
    layers.Dense(1)
  ])

  optimizer = tf.keras.optimizers.RMSprop(0.001)

  model.compile(loss='mean_squared_error',
                optimizer=optimizer,
                metrics=['mean_absolute_error', 'mean_squared_error'])
  return model

def build_model_B():
  model = keras.Sequential([
    layers.Dense(20, activation=tf.nn.relu, input_shape=[len(train.keys())]),
    layers.Dense(1)
  ])

  optimizer = tf.keras.optimizers.RMSprop(0.001)

  model.compile(loss='mean_squared_error',
                optimizer=optimizer,
                metrics=['mean_absolute_error', 'mean_squared_error'])
  return model
```

In [11]:
```
%%time
model_A = build_model_A()
model_A.summary()
model_B = build_model_B()
model_B.summary()
```

```
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/python/ops/resource_v
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/python/keras/utils/lo
Instructions for updating:
Use tf.cast instead.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 5)                 100
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 6
=================================================================
Total params: 106
Trainable params: 106
Non-trainable params: 0
_________________________________________________________________
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 20)                400
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 21
=================================================================
Total params: 421
Trainable params: 421
Non-trainable params: 0
_________________________________________________________________
CPU times: user 297 ms, sys: 19.3 ms, total: 316 ms
Wall time: 314 ms
```

```
In [12]: %%time
         #change batch size?
         #try early stop?
         # The patience parameter is the amount of epochs to check for improvement
         #early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)

         #history_object = model.fit(train, train_labels, epochs=100, verbose=2,
         #         validation_data=(validate, validate_labels), callbacks=[early_stop])

         history_object_A = model_A.fit(normed_train, train_labels, epochs=50, verbose=2,
                  validation_data=(normed_validate, validate_labels))
```

```
Train on 3982 samples, validate on 1327 samples
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.p
Instructions for updating:
Use tf.cast instead.
```

```
Epoch 1/50
 - 0s - loss: 819.9451 - mean_absolute_error: 28.4216 - mean_squared_error: 819.9451 - val_loss
Epoch 2/50
 - 0s - loss: 747.0297 - mean_absolute_error: 26.9810 - mean_squared_error: 747.0297 - val_loss
Epoch 3/50
 - 0s - loss: 679.0713 - mean_absolute_error: 25.3801 - mean_squared_error: 679.0714 - val_loss
Epoch 4/50
 - 0s - loss: 621.1198 - mean_absolute_error: 23.6669 - mean_squared_error: 621.1198 - val_loss
Epoch 5/50
 - 0s - loss: 573.6425 - mean_absolute_error: 22.1886 - mean_squared_error: 573.6425 - val_loss
Epoch 6/50
 - 0s - loss: 528.0163 - mean_absolute_error: 21.0834 - mean_squared_error: 528.0163 - val_loss
Epoch 7/50
 - 0s - loss: 474.8648 - mean_absolute_error: 19.8912 - mean_squared_error: 474.8648 - val_loss
Epoch 8/50
 - 0s - loss: 412.7359 - mean_absolute_error: 18.4580 - mean_squared_error: 412.7358 - val_loss
Epoch 9/50
 - 0s - loss: 343.4194 - mean_absolute_error: 16.7386 - mean_squared_error: 343.4194 - val_loss
Epoch 10/50
 - 0s - loss: 271.2246 - mean_absolute_error: 14.7493 - mean_squared_error: 271.2246 - val_loss
Epoch 11/50
 - 0s - loss: 202.7312 - mean_absolute_error: 12.5838 - mean_squared_error: 202.7312 - val_loss
Epoch 12/50
 - 0s - loss: 142.2559 - mean_absolute_error: 10.3800 - mean_squared_error: 142.2559 - val_loss
Epoch 13/50
 - 0s - loss: 95.2516 - mean_absolute_error: 8.1940 - mean_squared_error: 95.2516 - val_loss: 8
Epoch 14/50
 - 0s - loss: 62.6015 - mean_absolute_error: 6.3247 - mean_squared_error: 62.6015 - val_loss: 6
Epoch 15/50
 - 0s - loss: 44.0292 - mean_absolute_error: 4.9613 - mean_squared_error: 44.0292 - val_loss: 5
Epoch 16/50
 - 0s - loss: 35.3565 - mean_absolute_error: 4.2881 - mean_squared_error: 35.3565 - val_loss: 4
Epoch 17/50
 - 0s - loss: 30.2536 - mean_absolute_error: 3.9432 - mean_squared_error: 30.2536 - val_loss: 3
Epoch 18/50
 - 0s - loss: 26.2665 - mean_absolute_error: 3.6637 - mean_squared_error: 26.2665 - val_loss: 3
Epoch 19/50
 - 0s - loss: 23.6472 - mean_absolute_error: 3.4489 - mean_squared_error: 23.6472 - val_loss: 1
Epoch 20/50
 - 0s - loss: 20.5591 - mean_absolute_error: 3.2375 - mean_squared_error: 20.5591 - val_loss: 1
Epoch 21/50
 - 0s - loss: 17.8305 - mean_absolute_error: 3.0434 - mean_squared_error: 17.8305 - val_loss: 1
Epoch 22/50
 - 0s - loss: 15.9715 - mean_absolute_error: 2.8815 - mean_squared_error: 15.9715 - val_loss: 1
Epoch 23/50
 - 0s - loss: 14.0106 - mean_absolute_error: 2.7432 - mean_squared_error: 14.0106 - val_loss: 1
Epoch 24/50
 - 0s - loss: 12.5273 - mean_absolute_error: 2.6247 - mean_squared_error: 12.5273 - val_loss: 1
```

```
Epoch 25/50
 - 0s - loss: 11.3701 - mean_absolute_error: 2.5204 - mean_squared_error: 11.3701 - val_loss:
Epoch 26/50
 - 0s - loss: 10.5236 - mean_absolute_error: 2.4353 - mean_squared_error: 10.5236 - val_loss:
Epoch 27/50
 - 0s - loss: 9.7365 - mean_absolute_error: 2.3531 - mean_squared_error: 9.7365 - val_loss: 11
Epoch 28/50
 - 0s - loss: 9.0530 - mean_absolute_error: 2.2782 - mean_squared_error: 9.0530 - val_loss: 10
Epoch 29/50
 - 0s - loss: 8.5169 - mean_absolute_error: 2.2082 - mean_squared_error: 8.5169 - val_loss: 10
Epoch 30/50
 - 0s - loss: 8.0442 - mean_absolute_error: 2.1526 - mean_squared_error: 8.0442 - val_loss: 10
Epoch 31/50
 - 0s - loss: 7.6578 - mean_absolute_error: 2.0950 - mean_squared_error: 7.6578 - val_loss: 9.6
Epoch 32/50
 - 0s - loss: 7.3374 - mean_absolute_error: 2.0489 - mean_squared_error: 7.3374 - val_loss: 9.4
Epoch 33/50
 - 0s - loss: 7.0633 - mean_absolute_error: 2.0040 - mean_squared_error: 7.0633 - val_loss: 9.2
Epoch 34/50
 - 0s - loss: 6.8014 - mean_absolute_error: 1.9630 - mean_squared_error: 6.8014 - val_loss: 9.0
Epoch 35/50
 - 0s - loss: 6.5579 - mean_absolute_error: 1.9253 - mean_squared_error: 6.5579 - val_loss: 8.7
Epoch 36/50
 - 0s - loss: 6.3273 - mean_absolute_error: 1.8848 - mean_squared_error: 6.3273 - val_loss: 8.6
Epoch 37/50
 - 0s - loss: 6.1428 - mean_absolute_error: 1.8588 - mean_squared_error: 6.1428 - val_loss: 8.3
Epoch 38/50
 - 0s - loss: 5.9680 - mean_absolute_error: 1.8240 - mean_squared_error: 5.9680 - val_loss: 8.0
Epoch 39/50
 - 0s - loss: 5.8046 - mean_absolute_error: 1.8014 - mean_squared_error: 5.8046 - val_loss: 8.1
Epoch 40/50
 - 0s - loss: 5.6595 - mean_absolute_error: 1.7753 - mean_squared_error: 5.6595 - val_loss: 7.8
Epoch 41/50
 - 0s - loss: 5.5318 - mean_absolute_error: 1.7552 - mean_squared_error: 5.5318 - val_loss: 7.7
Epoch 42/50
 - 0s - loss: 5.3865 - mean_absolute_error: 1.7326 - mean_squared_error: 5.3865 - val_loss: 7.5
Epoch 43/50
 - 0s - loss: 5.2384 - mean_absolute_error: 1.7018 - mean_squared_error: 5.2384 - val_loss: 7.5
Epoch 44/50
 - 0s - loss: 5.1073 - mean_absolute_error: 1.6727 - mean_squared_error: 5.1073 - val_loss: 7.1
Epoch 45/50
 - 0s - loss: 4.9715 - mean_absolute_error: 1.6500 - mean_squared_error: 4.9715 - val_loss: 6.9
Epoch 46/50
 - 0s - loss: 4.8653 - mean_absolute_error: 1.6333 - mean_squared_error: 4.8653 - val_loss: 6.9
Epoch 47/50
 - 0s - loss: 4.7407 - mean_absolute_error: 1.6085 - mean_squared_error: 4.7407 - val_loss: 6.8
Epoch 48/50
 - 0s - loss: 4.6267 - mean_absolute_error: 1.5924 - mean_squared_error: 4.6267 - val_loss: 6.8
```

```
Epoch 49/50
 - 0s - loss: 4.5303 - mean_absolute_error: 1.5655 - mean_squared_error: 4.5303 - val_loss: 6.4
Epoch 50/50
 - 0s - loss: 4.4304 - mean_absolute_error: 1.5514 - mean_squared_error: 4.4304 - val_loss: 6.5
CPU times: user 10.7 s, sys: 3.03 s, total: 13.7 s
Wall time: 6.91 s
```

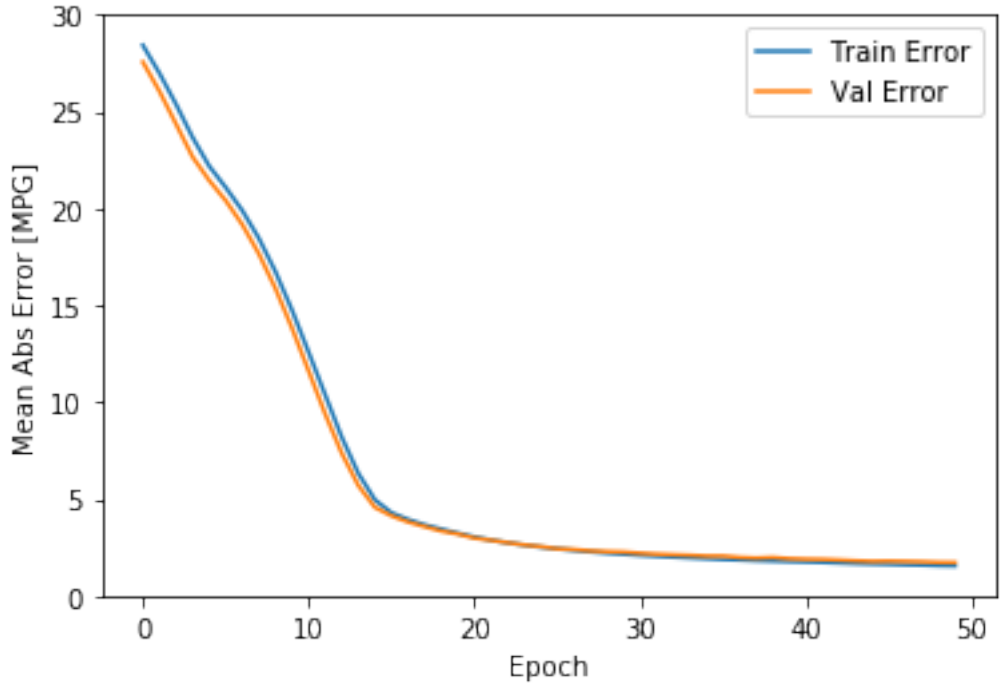
```
In [13]: %%time
         history_object_B = model_B.fit(normed_train, train_labels, epochs=50, verbose=2,
                  validation_data=(normed_validate, validate_labels))
```

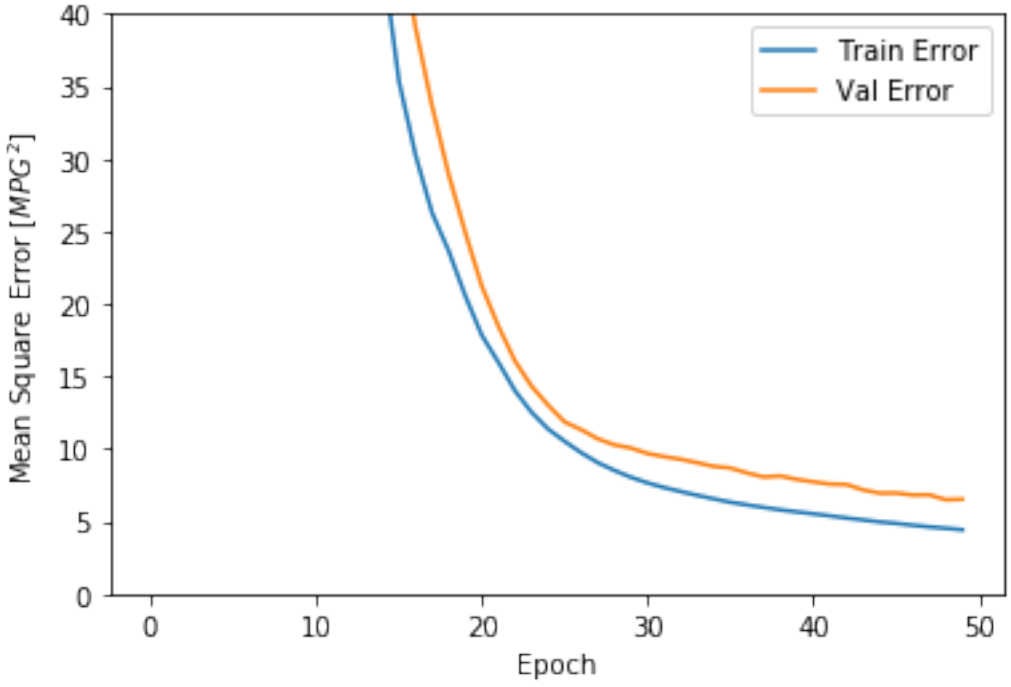
```
Train on 3982 samples, validate on 1327 samples
Epoch 1/50
 - 0s - loss: 773.9948 - mean_absolute_error: 27.6219 - mean_squared_error: 773.9949 - val_loss
Epoch 2/50
 - 0s - loss: 595.0250 - mean_absolute_error: 24.0686 - mean_squared_error: 595.0250 - val_loss
Epoch 3/50
 - 0s - loss: 413.3285 - mean_absolute_error: 19.5667 - mean_squared_error: 413.3284 - val_loss
Epoch 4/50
 - 0s - loss: 252.4994 - mean_absolute_error: 14.5458 - mean_squared_error: 252.4994 - val_loss
Epoch 5/50
 - 0s - loss: 140.8039 - mean_absolute_error: 9.8910 - mean_squared_error: 140.8039 - val_loss
Epoch 6/50
 - 0s - loss: 87.6197 - mean_absolute_error: 7.4378 - mean_squared_error: 87.6197 - val_loss: 8
Epoch 7/50
 - 0s - loss: 59.8197 - mean_absolute_error: 6.0619 - mean_squared_error: 59.8197 - val_loss: 5
Epoch 8/50
 - 0s - loss: 38.6836 - mean_absolute_error: 4.7490 - mean_squared_error: 38.6836 - val_loss: 3
Epoch 9/50
 - 0s - loss: 24.6896 - mean_absolute_error: 3.7451 - mean_squared_error: 24.6896 - val_loss: 2
Epoch 10/50
 - 0s - loss: 17.8148 - mean_absolute_error: 3.1844 - mean_squared_error: 17.8148 - val_loss: 2
Epoch 11/50
 - 0s - loss: 14.4613 - mean_absolute_error: 2.8676 - mean_squared_error: 14.4613 - val_loss: 1
Epoch 12/50
 - 0s - loss: 12.5121 - mean_absolute_error: 2.6599 - mean_squared_error: 12.5121 - val_loss: 1
Epoch 13/50
 - 0s - loss: 10.9223 - mean_absolute_error: 2.4934 - mean_squared_error: 10.9223 - val_loss: 1
Epoch 14/50
 - 0s - loss: 9.6090 - mean_absolute_error: 2.3507 - mean_squared_error: 9.6089 - val_loss: 14
Epoch 15/50
 - 0s - loss: 8.5949 - mean_absolute_error: 2.2304 - mean_squared_error: 8.5949 - val_loss: 13
Epoch 16/50
 - 0s - loss: 7.8226 - mean_absolute_error: 2.1352 - mean_squared_error: 7.8226 - val_loss: 12
Epoch 17/50
 - 0s - loss: 7.2379 - mean_absolute_error: 2.0517 - mean_squared_error: 7.2379 - val_loss: 12
Epoch 18/50
```

```
 - 0s - loss: 6.7453 - mean_absolute_error: 1.9783 - mean_squared_error: 6.7453 - val_loss: 11
Epoch 19/50
 - 0s - loss: 6.2872 - mean_absolute_error: 1.9116 - mean_squared_error: 6.2872 - val_loss: 11
Epoch 20/50
 - 0s - loss: 5.9011 - mean_absolute_error: 1.8506 - mean_squared_error: 5.9011 - val_loss: 10
Epoch 21/50
 - 0s - loss: 5.5664 - mean_absolute_error: 1.7973 - mean_squared_error: 5.5664 - val_loss: 10
Epoch 22/50
 - 0s - loss: 5.2865 - mean_absolute_error: 1.7505 - mean_squared_error: 5.2865 - val_loss: 10
Epoch 23/50
 - 0s - loss: 5.0292 - mean_absolute_error: 1.7092 - mean_squared_error: 5.0292 - val_loss: 9.9
Epoch 24/50
 - 0s - loss: 4.8014 - mean_absolute_error: 1.6672 - mean_squared_error: 4.8014 - val_loss: 9.6
Epoch 25/50
 - 0s - loss: 4.5948 - mean_absolute_error: 1.6293 - mean_squared_error: 4.5948 - val_loss: 9.5
Epoch 26/50
 - 0s - loss: 4.4061 - mean_absolute_error: 1.5941 - mean_squared_error: 4.4061 - val_loss: 9.4
Epoch 27/50
 - 0s - loss: 4.2370 - mean_absolute_error: 1.5602 - mean_squared_error: 4.2370 - val_loss: 9.0
Epoch 28/50
 - 0s - loss: 4.0667 - mean_absolute_error: 1.5275 - mean_squared_error: 4.0667 - val_loss: 8.9
Epoch 29/50
 - 0s - loss: 3.9241 - mean_absolute_error: 1.4996 - mean_squared_error: 3.9241 - val_loss: 8.8
Epoch 30/50
 - 0s - loss: 3.7957 - mean_absolute_error: 1.4731 - mean_squared_error: 3.7957 - val_loss: 8.6
Epoch 31/50
 - 0s - loss: 3.6645 - mean_absolute_error: 1.4403 - mean_squared_error: 3.6645 - val_loss: 8.4
Epoch 32/50
 - 0s - loss: 3.5596 - mean_absolute_error: 1.4153 - mean_squared_error: 3.5596 - val_loss: 8.1
Epoch 33/50
 - 0s - loss: 3.4504 - mean_absolute_error: 1.3971 - mean_squared_error: 3.4504 - val_loss: 8.1
Epoch 34/50
 - 0s - loss: 3.3678 - mean_absolute_error: 1.3733 - mean_squared_error: 3.3678 - val_loss: 7.9
Epoch 35/50
 - 0s - loss: 3.2635 - mean_absolute_error: 1.3524 - mean_squared_error: 3.2635 - val_loss: 8.0
Epoch 36/50
 - 0s - loss: 3.1758 - mean_absolute_error: 1.3237 - mean_squared_error: 3.1758 - val_loss: 7.8
Epoch 37/50
 - 0s - loss: 3.0931 - mean_absolute_error: 1.3069 - mean_squared_error: 3.0931 - val_loss: 7.9
Epoch 38/50
 - 0s - loss: 3.0115 - mean_absolute_error: 1.2850 - mean_squared_error: 3.0115 - val_loss: 7.8
Epoch 39/50
 - 0s - loss: 2.9227 - mean_absolute_error: 1.2572 - mean_squared_error: 2.9227 - val_loss: 7.7
Epoch 40/50
 - 0s - loss: 2.8595 - mean_absolute_error: 1.2413 - mean_squared_error: 2.8595 - val_loss: 7.6
Epoch 41/50
 - 0s - loss: 2.7811 - mean_absolute_error: 1.2226 - mean_squared_error: 2.7811 - val_loss: 7.8
Epoch 42/50
```

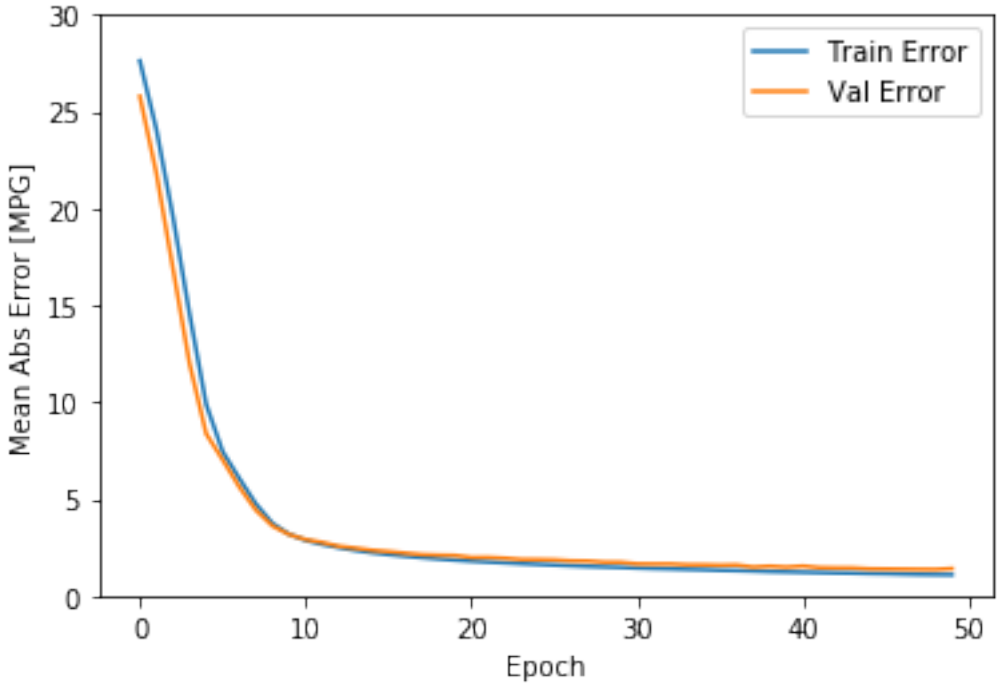
```
 - 0s - loss: 2.7062 - mean_absolute_error: 1.2037 - mean_squared_error: 2.7062 - val_loss: 7.8
Epoch 43/50
 - 0s - loss: 2.6558 - mean_absolute_error: 1.1872 - mean_squared_error: 2.6558 - val_loss: 7.4
Epoch 44/50
 - 0s - loss: 2.6156 - mean_absolute_error: 1.1758 - mean_squared_error: 2.6156 - val_loss: 7.5
Epoch 45/50
 - 0s - loss: 2.5571 - mean_absolute_error: 1.1576 - mean_squared_error: 2.5571 - val_loss: 7.4
Epoch 46/50
 - 0s - loss: 2.5156 - mean_absolute_error: 1.1467 - mean_squared_error: 2.5156 - val_loss: 7.5
Epoch 47/50
 - 0s - loss: 2.4696 - mean_absolute_error: 1.1272 - mean_squared_error: 2.4696 - val_loss: 7.5
Epoch 48/50
 - 0s - loss: 2.4222 - mean_absolute_error: 1.1109 - mean_squared_error: 2.4222 - val_loss: 7.3
Epoch 49/50
 - 0s - loss: 2.3941 - mean_absolute_error: 1.1015 - mean_squared_error: 2.3941 - val_loss: 7.4
Epoch 50/50
 - 0s - loss: 2.3431 - mean_absolute_error: 1.0935 - mean_squared_error: 2.3431 - val_loss: 7.7
CPU times: user 11 s, sys: 3.13 s, total: 14.1 s
Wall time: 7.18 s
```

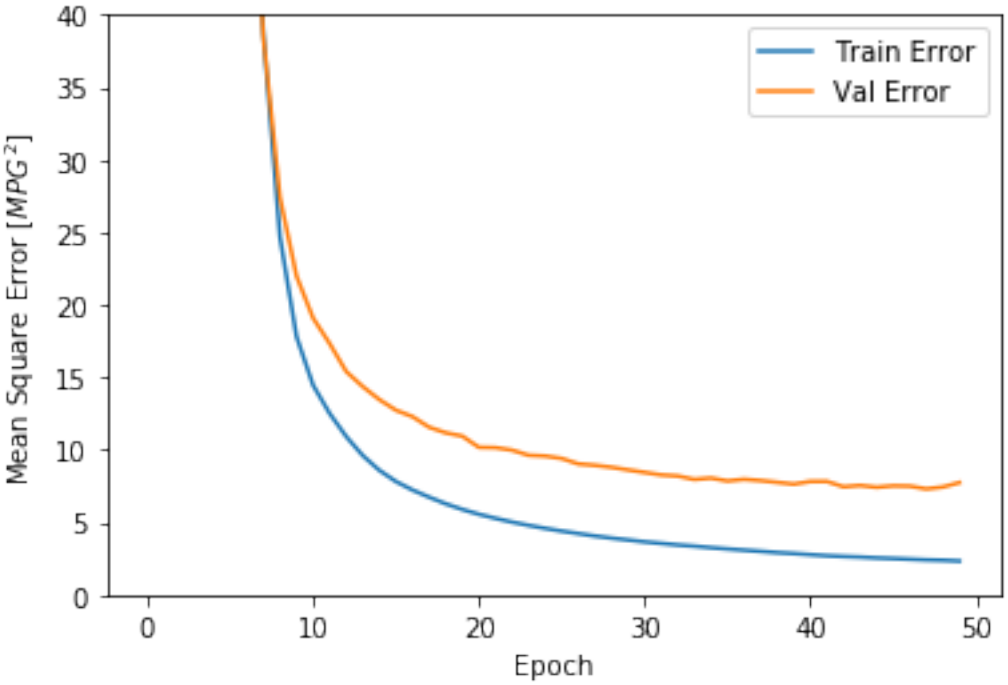
```
In [14]: def plot_history(history):
             hist = pd.DataFrame(history.history)
             hist['epoch'] = history.epoch

             plt.figure()
             plt.xlabel('Epoch')
             plt.ylabel('Mean Abs Error [MPG]')
             plt.plot(hist['epoch'], hist['mean_absolute_error'],
                      label='Train Error')
             plt.plot(hist['epoch'], hist['val_mean_absolute_error'],
                      label = 'Val Error')
             plt.ylim([0,30])
             plt.legend()

             plt.figure()
             plt.xlabel('Epoch')
             plt.ylabel('Mean Square Error [$MPG^2$]')
             plt.plot(hist['epoch'], hist['mean_squared_error'],
                      label='Train Error')
             plt.plot(hist['epoch'], hist['val_mean_squared_error'],
                      label = 'Val Error')
             plt.ylim([0,40])
             plt.legend()
             plt.show()

         plot_history(history_object_A)
```






```
In [15]: plot_history(history_object_B)
```


Train Error
Val Error
Mean Abs Error [MPG]
Epoch
0
5
10
15
20
25
30
0
10
20
30
40
50


Train Error
Val Error
Mean Square Error [MPG^2]
Epoch
0
5
10
15
20
25
30
35
40
0
10
20
30
40
50