databricksNET_regresion (one) (17)

Introduction

```
# Deprecated prior art with "tilted loss function":
https://towardsdatascience.com/deep-quantile-regression-c85481548b5a
```

Configuration

```
# Eager
import tensorflow as tf
tf.enable_eager_execution()
```

```
/databricks/python/lib/python3.7/site-packages/botocore/vendored/requests/packages/url
lib3/_collections.py:1: DeprecationWarning: Using or importing the ABCs from 'collecti
ons' instead of from 'collections.abc' is deprecated, and in 3.8 it will stop working
  from collections import Mapping, MutableMapping
```

```
# fix random seed for reproducibility
import random
import scipy
import numpy as np
import keras

RANDOM_SEED = 0
random.seed(RANDOM_SEED)
np.random.seed(RANDOM_SEED)
tf.set_random_seed(RANDOM_SEED)

tf.logging.set_verbosity(tf.logging.ERROR)
```

```
Using TensorFlow backend.
```

```
# Versioning
print("SCIPY=", scipy.__version__)
print("NUMPY=", np.__version__)
print("TENSORFLOW=", tf.__version__)
print("KERAS=", keras.__version__)
```

```
SCIPY= 1.2.1
NUMPY= 1.16.2
TENSORFLOW= 1.15.0
KERAS= 2.2.5
```

Credentials

```
# Get credential to read
key=dbutils.secrets.get(scope="application-secrets", key="enterprisedatalakeprodsas")

container_list = [
  'customer',
  'customerorder',
  'foundation',
  'pricing',
  'sales',
  'cdaworkspace'
]

# Setup access to read from any container in container list
for container in container_list:
  spark.conf.set(
    "fs.azure.sas.
{container}.enterprisedatalakeprod.blob.core.windows.net".format(container=container),
    key
  )

from pyspark.sql import SparkSession

# Create sparkContext
spark = SparkSession.builder.master("local[*]").getOrCreate()
sc = spark.sparkContext
sc._jsc.hadoopConfiguration().set("fs.azure.sas.cdaworkspace.enterprisedatalakeprod.bl
ob.core.windows.net", key)
```

Read

```
import json

def view_avro(view_name, file_location):
    df = spark.read.format('avro').option("inferSchema", "true").load(file_location)
    df.createOrReplaceTempView(view_name)
    shape = (df.count(), len(df.columns))
    print("DATAFRAME_SHAPE=", shape)
    return df, shape

def view_orc(view_name, file_location):
    df = spark.read.format('orc').option("inferSchema", "true").load(file_location)
    df.createOrReplaceTempView(view_name)
    shape = (df.count(), len(df.columns))
    print("DATAFRAME_SHAPE=", shape)
    return df, shape

def view_header_csv(view_name, file_location):
    df = spark.read.format('csv').option("inferSchema", "true").option("header",
'true').load(file_location)
    df.createOrReplaceTempView(view_name)
    shape = (df.count(), len(df.columns))
    print("DATAFRAME_SHAPE=", shape)
    return df, shape

def view_blob_csv(view_name, file_name):
    file_path =
'wasbs://cdaworkspace@enterprisedatalakeprod.blob.core.windows.net/cda/prod/'
    df = spark.read.format('csv').option("inferSchema", "true").option("header",
'true').load(file_path+file_name)
    df.createOrReplaceTempView(view_name)
    shape = (df.count(), len(df.columns))
    print("DATAFRAME_SHAPE=", shape)
    return df, shape

def view_schema_csv(view_name, file_location, table_schema):
  df = spark.read.format('csv').schema(table_schema).load(file_location)
  df.createOrReplaceTempView(view_name)
  shape = (df.count(), len(df.columns))
  print("DATAFRAME_SHAPE=", shape)
  return df, shape

def view_file(file_name):
  file_path =
'wasbs://cdaworkspace@enterprisedatalakeprod.blob.core.windows.net/cda/prod/'
  text_rdd = sc.textFile(file_path+file_name)
  return text_rdd
```

```
%fs ls
```

path
dbfs:/FileStore/
dbfs:/databricks/
dbfs:/databricks-datasets/
dbfs:/databricks-results/
dbfs:/delta/
dbfs:/ml/
dbfs:/mnt/
dbfs:/tmp/
dbfs:/user/

Training Dataset

Purchase

```
purchase_regression_df, purchase_regression_shape =
view_blob_csv('purchase_regression_view', 'LTV_BF_US_Regression_sales.csv')
```

```
DATAFRAME_SHAPE= (6492696, 102)
```

```
purchase_regression_df.printSchema()
```

```
root
 |-- customer_key: decimal(9,0) (nullable = true)
 |-- spend_6mo_sls: double (nullable = true)
 |-- repeat_spend_6mo_sls: double (nullable = true)
 |-- item_qty_6mo_sls: string (nullable = true)
 |-- spend_12mo_sls: double (nullable = true)
 |-- repeat_spend_12mo_sls: double (nullable = true)
 |-- item_qty_12mo_sls: string (nullable = true)
 |-- spend_24mo_sls: double (nullable = true)
 |-- repeat_spend_24mo_sls: double (nullable = true)
 |-- item_qty_24mo_sls: string (nullable = true)
 |-- onsale_qty_6mo_sls: string (nullable = true)
 |-- onsale_qty_12mo_sls: string (nullable = true)
 |-- onsale_qty_24mo_sls: string (nullable = true)
 |-- num_txns_6mo_sls: string (nullable = true)
 |-- num_txns_12mo_sls: string (nullable = true)
 |-- num_txns_24mo_sls: string (nullable = true)
 |-- repeat_num_txns_6mo_sls: string (nullable = true)
 |-- repeat_num_txns_12mo_sls: string (nullable = true)
 |-- repeat_num_txns_24mo_sls: string (nullable = true)
```

```
display(purchase_regression_df.describe())
```

summary	customer_key	spend_6mo_sls	repeat_spend_6mo_sls	item_qty_6mo_sls
count	6492696	6492696	6492696	6492696
mean	195510649.6123	47.991817372949725	2.3076863278369153	5.68607292861858
stddev	1.0857491852506673E8	108.26365583092777	40.53191933238156	9.123645379971963
min	115	-15.49	-3116.709999999994	1
max	395764197	48379.44000000747	11595.829999999876	NA

Return

```
return_regression_df, return_regression_shape =
view_blob_csv('return_regression_view', 'LTV_BF_US_Regression_return.csv')
```

```
DATAFRAME_SHAPE= (6492696, 46)
```

```
return_regression_df.printSchema()
```

```
root
 |-- customer_key: decimal(9,0) (nullable = true)
 |-- spend_6mo_rtn: double (nullable = true)
 |-- item_qty_6mo_rtn: string (nullable = true)
 |-- spend_12mo_rtn: double (nullable = true)
 |-- item_qty_12mo_rtn: string (nullable = true)
 |-- spend_24mo_rtn: double (nullable = true)
 |-- item_qty_24mo_rtn: string (nullable = true)
 |-- onsale_qty_6mo_rtn: string (nullable = true)
 |-- onsale_qty_12mo_rtn: string (nullable = true)
 |-- onsale_qty_24mo_rtn: string (nullable = true)
 |-- num_txns_6mo_rtn: string (nullable = true)
 |-- num_txns_12mo_rtn: string (nullable = true)
 |-- num_txns_24mo_rtn: string (nullable = true)
 |-- spend_6mo_men_rtn: double (nullable = true)
 |-- item_qty_6mo_men_rtn: string (nullable = true)
 |-- spend_6mo_women_rtn: double (nullable = true)
 |-- item_qty_6mo_women_rtn: string (nullable = true)
 |-- spend_6mo_accessories_rtn: double (nullable = true)
 |-- item_qty_6mo_accessories_rtn: string (nullable = true)
```

```
display(return_regression_df.describe())
```

summary	customer_key	spend_6mo_rtn	item_qty_6mo_rtn	spend_12mo_rtn	iten
count	6492696	6492696	6492696	6492696	649
mean	195510649.6123	4.708404126731888	-2.980244666635949	8.40134604177409	-3.3
stddev	1.0857491852506663E8	29.0342673897853	3.9007880112960986	43.00482727367231	4.9
min	115	0.0	-1	0.0	-1
max	395764197	6640.179999999973	NA	9865.57999999996	NA

Labels

```
ground_truth_col_names = ['validation_spend_sls', 'validation_num_txns_sls',
'validation_item_qty_sls',
                    'validation_spend_rtn', 'validation_num_txns_rtn',
'validation_item_qty_rtn',
                    'net_txn_amt']
purchase_label_name = 'validation_spend_sls'
return_label_name = 'validation_spend_rtn'
```

Join Purchase and Return

```
to_join_cols = ['customer_key', return_label_name]
return_regression_select_df = return_regression_df.select(*to_join_cols)


overall_df = purchase_regression_df.join(return_regression_select_df,
on='customer_key', how='inner')


overall_df.printSchema()
```

```
root
 |-- customer_key: decimal(9,0) (nullable = true)
 |-- spend_6mo_sls: double (nullable = true)
 |-- repeat_spend_6mo_sls: double (nullable = true)
 |-- item_qty_6mo_sls: string (nullable = true)
 |-- spend_12mo_sls: double (nullable = true)
 |-- repeat_spend_12mo_sls: double (nullable = true)
 |-- item_qty_12mo_sls: string (nullable = true)
 |-- spend_24mo_sls: double (nullable = true)
 |-- repeat_spend_24mo_sls: double (nullable = true)
 |-- item_qty_24mo_sls: string (nullable = true)
 |-- onsale_qty_6mo_sls: string (nullable = true)
 |-- onsale_qty_12mo_sls: string (nullable = true)
 |-- onsale_qty_24mo_sls: string (nullable = true)
 |-- num_txns_6mo_sls: string (nullable = true)
 |-- num_txns_12mo_sls: string (nullable = true)
 |-- num_txns_24mo_sls: string (nullable = true)
 |-- repeat_num_txns_6mo_sls: string (nullable = true)
 |-- repeat_num_txns_12mo_sls: string (nullable = true)
 |-- repeat_num_txns_24mo_sls: string (nullable = true)
 |-- spend_6mo_men_sls: double (nullable = true)
```

```
display(overall_df)
```

customer_key	spend_6mo_sls	repeat_spend_6mo_sls	item_qty_6mo_sls	spend_12mo_sl
8440	73.77999999999999	0	4	73.77999999999
13248	0	0	NA	0
31156	23.39	0	2	23.39
32912	171.78000000000003	0	8	267.2500000000
33013	0	0	NA	0

39473	16	0	1	16
40436	0	0	NA	0
55474	19.98	0	2	19.98

Showing the first 1000 rows.

Model Training

```
import warnings
warnings.filterwarnings('ignore')

# General libraries
import sys,os,time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import seaborn as sb
from statistics import median

# ML libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.pipeline import Pipeline
from sklearn.utils import resample
from sklearn.ensemble import RandomForestRegressor as rfr
from keras import layers
import tensorflow as tf
from keras import backend as K
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn import preprocessing
from keras import regularizers
from keras.layers import Dropout
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from keras.constraints import max_norm
from keras.models import model_from_json
from keras.layers.advanced_activations import LeakyReLU
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import median_absolute_error
from sklearn.preprocessing import MinMaxScaler
import keras.backend as K
```

```
from pyspark.sql.types import DoubleType
import pyspark.sql.functions as F


CAST_COLUMN_TAG = '_'

def na_and_fill(df):
  return df \
        .dropna()
         # TODO: revisit this assumption vs fillna(0)?

def cast_column(df):
  for col_name in df_in.columns:
    cast_col_name = col_name + CAST_COLUMN_TAG
    df = df \
          .withColumn(cast_col_name, df[col_name].cast(DoubleType())) \
          .drop(col_name)
  return df

def drop_null(df):
  null_counts = df.select([F.count(F.when(F.col(c).isNull(), c)).alias(c) for c in
df.columns]) \
                      .collect()[0] \
                      .asDict()
  to_drop = [k for k, v in null_counts.items() if v > 0]
  df = df.drop(*to_drop)

  return df, to_drop

# TODO: Cap outliers?

def filter_negative(df):
  for col_name in ground_truth_col_names:
    try:
      df = df.filter(df[col_name] > 0)
    except:
      pass
  return df
```

```
df_in = overall_df.drop('customer_key')

df = na_and_fill(df_in)
df = cast_column(df)
df = filter_negative(df)
df_final, to_drop = drop_null(df)

print("COLUMN_TO_DROP=", to_drop)

assert df_final.count() > 0, "DATAFRAME_LENGTH_ERROR"
```

```
COLUMN_TO_DROP= ['item_qty_6mo_sls_', 'item_qty_12mo_sls_', 'item_qty_24mo_sls_', 'ons
ale_qty_6mo_sls_', 'onsale_qty_12mo_sls_', 'onsale_qty_24mo_sls_', 'num_txns_6mo_sls_'
, 'num_txns_12mo_sls_', 'num_txns_24mo_sls_', 'repeat_num_txns_6mo_sls_', 'repeat_num_
txns_12mo_sls_', 'repeat_num_txns_24mo_sls_', 'item_qty_6mo_men_sls_', 'item_qty_6mo_w
omen_sls_', 'item_qty_6mo_accessories_sls_', 'item_qty_6mo_plcb_sls_', 'item_qty_12mo_
plcb_sls_', 'item_qty_24mo_plcb_sls_', 'num_plcb_txns_6mo_sls_', 'num_plcb_txns_12mo_s
ls_', 'num_plcb_txns_24mo_sls_', 'item_qty_6mo_sls_sb_', 'item_qty_12mo_sls_sb_', 'ite
m_qty_24mo_sls_sb_', 'onsale_qty_6mo_sls_sb_', 'onsale_qty_12mo_sls_sb_', 'onsale_qty_
24mo_sls_sb_', 'num_txns_6mo_sls_sb_', 'num_txns_12mo_sls_sb_', 'num_txns_24mo_sls_sb_
', 'item_qty_plcb_6mo_sls_sb_', 'item_qty_plcb_12mo_sls_sb_', 'item_qty_plcb_24mo_sls_
sb_', 'num_plcb_txns_6mo_sls_sb_', 'num_plcb_txns_12mo_sls_sb_', 'num_plcb_txns_24mo_s
ls_sb_', 'item_qty_6mo_onl_sls_', 'item_qty_12mo_onl_sls_', 'item_qty_24mo_onl_sls_',
'num_txns_6mo_onl_sls_', 'num_txns_12mo_onl_sls_', 'num_txns_24mo_onl_sls_', 'item_qty
_6mo_rtl_sls_', 'item_qty_12mo_rtl_sls_', 'item_qty_24mo_rtl_sls_', 'num_txns_6mo_rtl_
sls_', 'num_txns_12mo_rtl_sls_', 'num_txns_24mo_rtl_sls_', 'stores_shopped_6mo_sls_',
'stores_shopped_12mo_sls_', 'stores_shopped_24mo_sls_', 'customer_duration_sb_', 'days
_first_pur_sb_', 'days_last_pur_sb_', 'customer_duration_sb_onl_', 'days_first_pur_sb_
onl_', 'days_last_pur_sb_onl_', 'customer_duration_sb_rtl_', 'days_first_pur_sb_rtl_',
'days_last_pur_sb_rtl_', 'customer_duration_', 'days_first_pur_', 'days_last_pur_', 'c
ustomer_duration_onl_', 'days_first_pur_onl_', 'days_last_pur_onl_', 'customer_duratio
n_rtl_', 'days_first_pur_rtl_', 'days_last_pur_rtl_', 'days_on_books_', 'card_status_'
, 'mapped_source_', 'validation_item_qty_sls_', 'validation_num_txns_sls_', 'segment_f
lags_']
```

```
train_df, test_df, val_df = df_final.randomSplit([.7, .25, .05])
train_df.cache()
test_df.cache()
val_df.cache()

train_df.count(), test_df.count(), val_df.count()
```

```
Out[21]: (4544001, 1623432, 325263)
```

```
display(train_df)
```

spend_6mo_sls_	repeat_spend_6mo_sls_	spend_12mo_sls_	repeat_spend_12mo_sls_	spen
0	-16.450000000000017	0	-16.450000000000017	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Showing the first 1000 rows.

Ground truth columns renamed

```
ground_truth_col_renamed = [x+CAST_COLUMN_TAG for x in ground_truth_col_names]
purchase_label_renamed = purchase_label_name + CAST_COLUMN_TAG
return_label_renamed = return_label_name + CAST_COLUMN_TAG
```

Input pipeline

```
# https://dwgeek.com/python-pyspark-iterator-how-to-create-and-use.html/
def get_dict_iter(df):
  return df.rdd.map(lambda r: r.asDict()).toLocalIterator()


# test
sample_iter = get_dict_iter(val_df)
```

```
def get_df_batch(df_iter, batch_size=100):
  batch = []
  for i in range(batch_size):
    try:
      next_is = next(df_iter)
      batch.append(next_is)
      # print(next_is)
    except:
      pass
  return batch

def get_tf_dataset_batch(df_iter, batch_size):
  batch_dict = get_df_batch(df_iter, batch_size)
  batch_pd = pd.DataFrame.from_dict(batch_dict)

  purchase_label_col = batch_pd.pop(purchase_label_renamed)
  return_label_col = batch_pd.pop(return_label_renamed)
  net_label_col = purchase_label_col - return_label_col
  dict_labels = (dict(batch_pd), net_label_col)

  tf_dataset = tf.data.Dataset.from_tensor_slices(dict_labels)
  return tf_dataset.batch(batch_size), batch_pd


# test
sample_ds, sample_pd = get_tf_dataset_batch(sample_iter, batch_size=1)

for feature_batch, label_batch in sample_ds.take(1):
  print('EVERY_FEATURE=', list(feature_batch.keys()))
  print('FEATURE_BATH repeat_spend_12mo_sls_=',
feature_batch['repeat_spend_12mo_sls_'])
  print('TARGET_BATCH=', label_batch )
```

```
EVERY_FEATURE= ['net_txn_amt_', 'repeat_spend_12mo_sls_', 'repeat_spend_24mo_sls_', 'r
epeat_spend_6mo_sls_', 'spend_12mo_onl_sls_', 'spend_12mo_plcb_sls_', 'spend_12mo_plcb
_sls_sb_', 'spend_12mo_rtl_sls_', 'spend_12mo_sls_', 'spend_12mo_sls_sb_', 'spend_24mo
_onl_sls_', 'spend_24mo_plcb_sls_', 'spend_24mo_plcb_sls_sb_', 'spend_24mo_rtl_sls_',
'spend_24mo_sls_', 'spend_24mo_sls_sb_', 'spend_6mo_accessories_sls_', 'spend_6mo_men_
sls_', 'spend_6mo_onl_sls_', 'spend_6mo_plcb_sls_', 'spend_6mo_plcb_sls_sb_', 'spend_6
mo_rtl_sls_', 'spend_6mo_sls_', 'spend_6mo_sls_sb_', 'spend_6mo_women_sls_']
FEATURE_BATH repeat_spend_12mo_sls_= tf.Tensor([0.], shape=(1,), dtype=float64)
TARGET_BATCH= tf.Tensor([0.], shape=(1,), dtype=float64)
```

```
# test
from tensorflow import feature_column

sample_batch = next(iter(sample_ds))[0]
col = feature_column.numeric_column('repeat_spend_12mo_sls_')
sample_layer = tf.keras.layers.DenseFeatures(col)
layer_numpy = sample_layer(sample_batch).numpy()

print("layer_numpy=", layer_numpy)
assert len(layer_numpy)>0, "BATCHING_FAILED"
```

```
layer_numpy= [[0.]]
```

```
from tensorflow.keras.layers import Layer

def avoid_snooping_truth(xy_pd):
  x_pd = xy_pd

  for col in ground_truth_col_renamed:
    try:
      x_pd.drop(col)
    except:
      pass

  return x_pd


def get_numeric_features(x_pd):
  feature_columns = []
  numeric_columns = x_pd.select_dtypes(include=['float64']).columns
  for header in numeric_columns:
    feature_columns.append(feature_column.numeric_column(header))
  return feature_columns


def get_features(xy_pd):
  x_pd = avoid_snooping_truth(xy_pd)

  feature_columns = []
  feature_columns += get_numeric_features(x_pd)
  # feature_columns += get_indicator_features(x_pd)

  feature_layer = tf.keras.layers.DenseFeatures(feature_columns)
  # print("feature_columns", feature_columns)
  return feature_layer


def get_indicator_features(x_pd):
  feature_columns = []
  indicator_columns = x_pd.select_dtypes(include=['object']).columns

  for col in indicator_columns:
    category_feature = feature_column.categorical_column_with_vocabulary_list(col,
indicator_columns)
    onehot_feature = feature_column.indicator_column(category_feature)
    feature_columns.append(onehot_feature)
  return feature_columns
```

```
# test
sample_layer = get_features(sample_pd)

sample_layer
```

```
Out[30]: <tensorflow.python.feature_column.dense_features.DenseFeatures at 0x7efd0e118
5f8>
```

```
def neural_net_model(input_layer,
                     hidden_layer_neuron_count=256, num_hidden_layers=4):
    layers = [input_layer]
    for i in range(num_hidden_layers):
      layers.append(tf.keras.layers.Dense(hidden_layer_neuron_count,
kernel_initializer='normal', activation='relu'))
    layers.append(tf.keras.layers.Dense(1, kernel_initializer='normal',
activation='linear'))
    model = tf.keras.Sequential(layers)
    return model
```

```
from collections import defaultdict

from keras.callbacks import History
from keras.callbacks import EarlyStopping


def train_model(model_key,
                loss_function,
                train_df, test_df,
                learning_rate,
                layer_neurons_nonlinear, num_layers_nonlinear,
                whole_epochs=1, epoch_per_batch=1000 # relies on EarlyStopping
               ):

  num_batches = 10
  batch_size = int(train_df.count()/num_batches)

  test_iter = get_dict_iter(test_df)
  test_ds, test_pd = get_tf_dataset_batch(test_iter, batch_size)
  feature_layer = get_features(test_pd)

  training_model = neural_net_model(feature_layer, layer_neurons_nonlinear,
num_layers_nonlinear)
  # keras_wrapper = keras.wrappers.scikit_learn.KerasRegressor(training_model)
  training_model.compile(loss=loss_function,
                optimizer=tf.keras.optimizers.SGD(learning_rate=learning_rate),
                metrics=[loss_function, 'mean_absolute_error'])

  training_history = []

  while whole_epochs > 0:
    train_iter = get_dict_iter(train_df) # .orderBy(rand()

    while num_batches > 0:
      print("\n\n*******************\n",
            "MODEL_KEY=", model_key,
            "REMAINING_EPOCHS=", whole_epochs,
            "REMAINING_BATCHES=", num_batches,
            "\n*******************\n")

      train_ds, train_pd = get_tf_dataset_batch(train_iter, batch_size)

      history = History()
      training_model.fit(train_ds,
                validation_data=test_ds,
                shuffle=True,
                epochs=epoch_per_batch,
                callbacks=[history,
                           EarlyStopping(monitor='val_loss', mode='auto',
patience=5)])  # min_delta=1
      training_history.append(history)

      num_batches-=1
    whole_epochs-=1

  return training_model, training_history


def train_models(hyper_params):
  param_model = {}
  param_history = {}

  for loss in hyper_params['loss_function']:
    for height in hyper_params['layer_neurons_nonlinear']:
      for depth in hyper_params['num_layers_nonlinear']:
        for rate in hyper_params['learning_rate']:
          for whole_epochs in hyper_params['whole_dataset_epochs']:
            for epoch_per_batch in hyper_params['epoch_per_batch']:
              try:
                model_key = '-' + 'depth=' + str(depth) + '-' + 'height=' +
str(height) +  \
                  '-' + 'whole_epochs=' + str(whole_epochs) + '-' + 'epoch_per_batch='
+ str(epoch_per_batch) + \
                  '-' + 'loss=' + loss + '-' + 'rate=' + str(rate)
                model, history = train_model(model_key,
                                             loss,
                                             train_df, test_df,
                                             rate,
                                             layer_neurons_nonlinear=height,
num_layers_nonlinear=depth,
                                             whole_epochs=whole_epochs,
epoch_per_batch=epoch_per_batch)
                param_model[model_key] = model
                param_history[model_key] = history
              except Exception as e:
                print("TRAINING_EXCEPTION", model_key, e)

  return param_model, param_history
```

```
def choose_model(param_model, param_history):
  COMPARISON_BATCH_SIZE = 250000
  test_iter = get_dict_iter(test_df)
  test_ds, test_pd = get_tf_dataset_batch(test_iter, batch_size=COMPARISON_BATCH_SIZE)

  chosen_key = None
  chosen_model = None
  chosen_test_loss = None
  chosen_test_error = None
  for model_key, model in param_model.items():
    try:
      test_evaluation = model.evaluate(test_ds)
      test_loss = test_evaluation[0]
      test_objective_error = test_evaluation[1]
      print("MODEL_EVALUATED=", model_key, "TRAIN_LOSS=", test_loss, "TRAIN_ERROR=",
test_objective_error)
      if chosen_model == None or (test_loss < chosen_test_loss):
        chosen_key = model_key
        chosen_model = model
        chosen_test_loss = test_loss
        chosen_test_error = test_objective_error
        print("MODEL_SELECTED=", model_key)
    except Exception as e:
      print(model_key, "FAILED_SELECTION", e)
      pass
  return chosen_key, chosen_model, chosen_test_loss, chosen_test_error


def measure_model_performance(chosen_key, chosen_model):
  try:
    MEASUREMENT_BATCH_SIZE = 50000
    val_iter = get_dict_iter(val_df)
    val_ds, val_pd = get_tf_dataset_batch(val_iter, batch_size=MEASUREMENT_BATCH_SIZE)

    val_evaluation = chosen_model.evaluate(val_ds)
    loss = val_evaluation[0]
    error = val_evaluation[1]
    print("CHOSEN_MODEL=", chosen_key, "VALIDATION_ERROR=", error)
  except Exception as e:
    print(e)
    pass
```

Run Model Training and Selection

Loss = mean_absolute_percentage_error

```
hyper_params = {}
hyper_params['loss_function'] = ['mean_absolute_percentage_error'] #
'mean_squared_error', mean_absolute_error, mean_squared_logarithmic_error,
cosine_similarity, huber_loss, log_cosh
hyper_params['num_layers_nonlinear'] = [1, 2, 3]
hyper_params['layer_neurons_nonlinear'] = [6, 12, 18]
hyper_params['learning_rate'] = [10**-12]

hyper_params['whole_dataset_epochs'] = [1] # 10
hyper_params['epoch_per_batch'] = [5]
```

```
param_model, param_history = train_models(hyper_params)
```

```
*******************
 MODEL_KEY= -depth=1-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_per
centage_error-rate=1e-12 REMAINING_EPOCHS= 1 REMAINING_BATCHES= 10
*******************

Epoch 1/5

      1/Unknown - 18s 18s/step - loss: 1945880192.0000 - mean_absolute_percentage_err
or: 1945880192.0000 - mean_absolute_error: 49.3841
1/1 [==============================] - 35s 35s/step - loss: 1945880192.0000 - mean_ab
solute_percentage_error: 1945880192.0000 - mean_absolute_error: 49.3841 - val_loss: 0
.0000e+00 - val_mean_absolute_percentage_error: 0.0000e+00 - val_mean_absolute_error:
0.0000e+00
Epoch 2/5

1/1 [==============================] - 32s 32s/step - loss: 262421216.0000 - mean_abs
olute_percentage_error: 262421216.0000 - mean_absolute_error: 46.2176 - val_loss: 330
273888.0000 - val_mean_absolute_percentage_error: 330273888.0000 - val_mean_absolute_
error: 45.5441
```

```
chosen_key, chosen_model, chosen_test_loss, chosen_test_error =
choose_model(param_model, param_history)
```

```
1/1 [==============================] - 32s 32s/step - loss: 8739077.0000 - mean_absolu
te_percentage_error: 8739077.0000 - mean_absolute_error: 45.5943
MODEL_EVALUATED= -depth=1-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute
_percentage_error-rate=1e-12 TRAIN_LOSS= 8739077.0 TRAIN_ERROR= 8739077.0
MODEL_SELECTED= -depth=1-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_
percentage_error-rate=1e-12

1/1 [==============================] - 9s 9s/step - loss: 8143327.0000 - mean_absolute
_percentage_error: 8143327.0000 - mean_absolute_error: 45.6253
MODEL_EVALUATED= -depth=2-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute
_percentage_error-rate=1e-12 TRAIN_LOSS= 8143327.0 TRAIN_ERROR= 8143327.0
MODEL_SELECTED= -depth=2-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_
percentage_error-rate=1e-12

1/1 [==============================] - 8s 8s/step - loss: 3672998.5000 - mean_absolute
_percentage_error: 3672998.5000 - mean_absolute_error: 45.6250
MODEL_EVALUATED= -depth=3-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute
_percentage_error-rate=1e-12 TRAIN_LOSS= 3672998.5 TRAIN_ERROR= 3672998.5
MODEL_SELECTED= -depth=3-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_
percentage_error-rate=1e-12

1/1 [==============================] - 9s 9s/step - loss: 72781640.0000 - mean_absolut
e_percentage_error: 72781640.0000 - mean_absolute_error: 45.7747
MODEL_EVALUATED= -depth=1-height=12-whole_epochs=1-epoch_per_batch=5-loss=mean_absolut
e_percentage_error-rate=1e-12 TRAIN_LOSS= 72781640.0 TRAIN_ERROR= 72781640.0

1/1 [==============================] - 8s 8s/step - loss: 10194882.0000 - mean_absolut
e_percentage_error: 10194882.0000 - mean_absolute_error: 45.6218
MODEL_EVALUATED= -depth=2-height=12-whole_epochs=1-epoch_per_batch=5-loss=mean_absolut
e_percentage_error-rate=1e-12 TRAIN_LOSS= 10194882.0 TRAIN_ERROR= 10194882.0

1/1 [==============================] - 9s 9s/step - loss: 6754391.0000 - mean_absolute
_percentage_error: 6754391.0000 - mean_absolute_error: 45.6325
MODEL_EVALUATED= -depth=3-height=12-whole_epochs=1-epoch_per_batch=5-loss=mean_absolut
e_percentage_error-rate=1e-12 TRAIN_LOSS= 6754391.0 TRAIN_ERROR= 6754391.0

1/1 [==============================] - 9s 9s/step - loss: 132566992.0000 - mean_absolu
te_percentage_error: 132566992.0000 - mean_absolute_error: 45.5887
MODEL_EVALUATED= -depth=1-height=18-whole_epochs=1-epoch_per_batch=5-loss=mean_absolut
e_percentage_error-rate=1e-12 TRAIN_LOSS= 132566992.0 TRAIN_ERROR= 132566990.0
```

```
1/1 [==============================] - 9s 9s/step - loss: 42452596.0000 - mean_absolut
e_percentage_error: 42452596.0000 - mean_absolute_error: 45.6428
MODEL_EVALUATED= -depth=2-height=18-whole_epochs=1-epoch_per_batch=5-loss=mean_absolut
e_percentage_error-rate=1e-12 TRAIN_LOSS= 42452596.0 TRAIN_ERROR= 42452596.0

1/1 [==============================] - 9s 9s/step - loss: 11508606.0000 - mean_absolut
e_percentage_error: 11508606.0000 - mean_absolute_error: 45.6334
MODEL_EVALUATED= -depth=3-height=18-whole_epochs=1-epoch_per_batch=5-loss=mean_absolut
e_percentage_error-rate=1e-12 TRAIN_LOSS= 11508606.0 TRAIN_ERROR= 11508606.0
```

```
measure_model_performance(chosen_key, chosen_model)
```

```
1/1 [==============================] - 2s 2s/step - loss: 3666583.2500 - mean_absolute
_percentage_error: 3666583.2500 - mean_absolute_error: 46.6171
CHOSEN_MODEL= -depth=3-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_pe
rcentage_error-rate=1e-12 VALIDATION_ERROR= 3666583.2
```

Plot

Plot Helper Classes

```
from collections import defaultdict


class model_history():

  def __init__(self):
    self.loss_history = defaultdict(list)
    self.error_history = defaultdict(list)
    self.percentage_error_history = defaultdict(list)

  def add(self, key, values):
    if key in ["loss", "val_loss"]:
      self.loss_history[key].extend(values)
    if key in ["mean_absolute_error", "val_mean_absolute_error"]:
      self.error_history[key].extend(values)
    if key in ["mean_absolute_percentage_error",
"val_mean_absolute_percentage_error"]:
      self.percentage_error_history[key].extend(values)

# test
h = model_history()
h.add('loss', [1,2,3])
h.add('loss', [3,2,1])
h.loss_history
```

```
Out[40]: defaultdict(list, {'loss': [1, 2, 3, 3, 2, 1]})
```

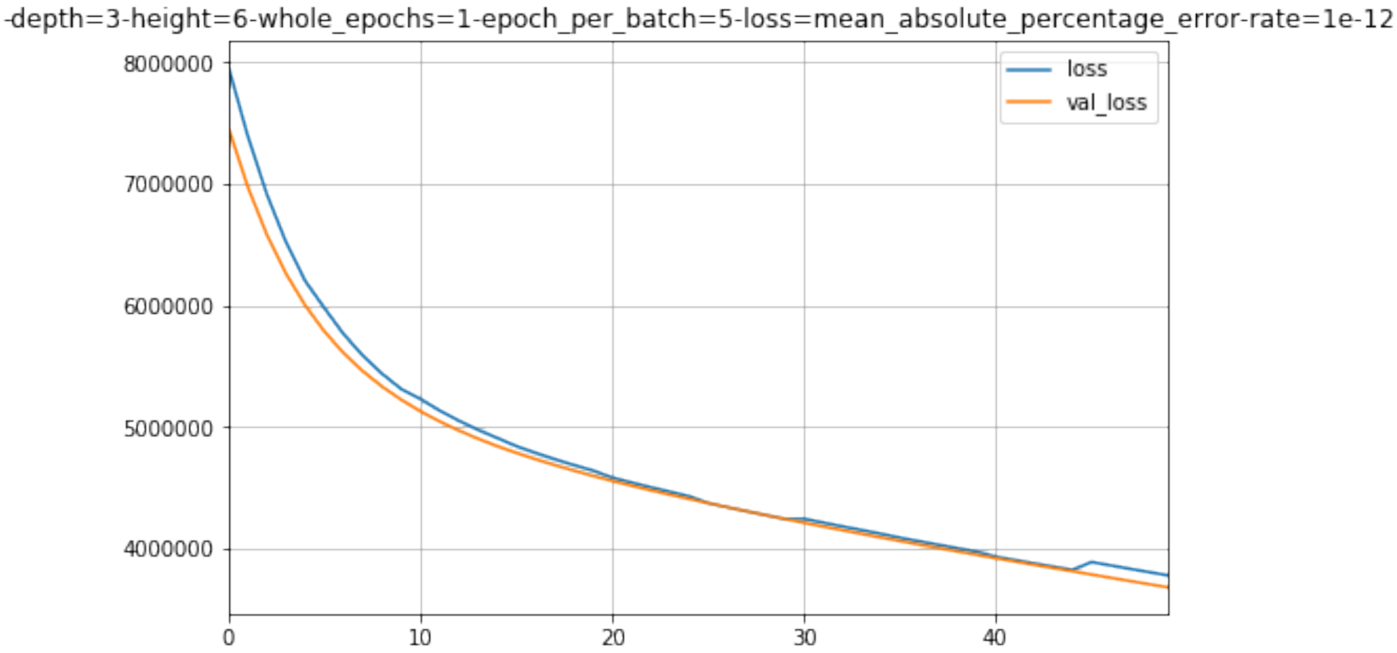
```
model_train_history = {}

for model_key in param_history.keys():
  try:
    history_arr = param_history[model_key]
    for batch_id in range(len(history_arr)):
      history_batch = history_arr[batch_id]
      # print(history_batch.history.keys())
      for metric_name, metric_history in history_batch.history.items():
        if model_key not in model_train_history:
          this_history = model_history()
        else:
          this_history = model_train_history[model_key]
        this_history.add(metric_name, metric_history)
        model_train_history[model_key] = this_history
  except:
    pass
```

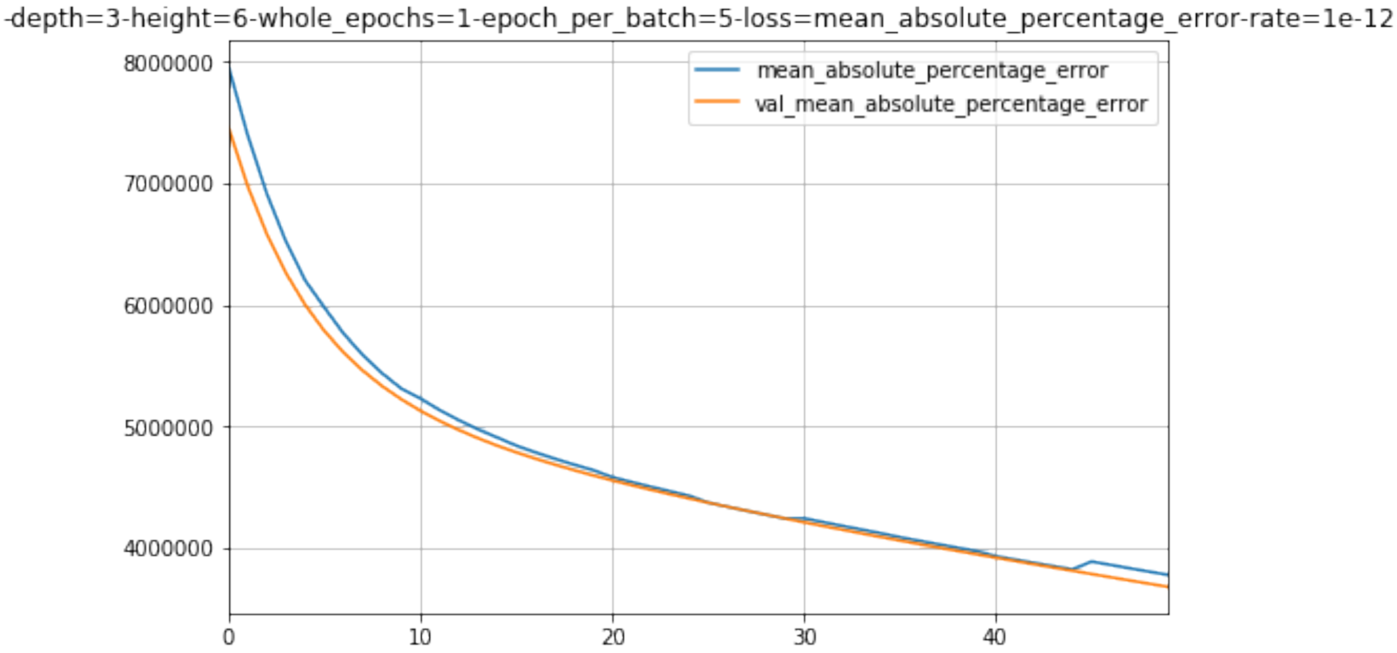
```
def plot_learning_curves(model_key, train_history):
  try:
    print("model_key=", model_key)
    pd.DataFrame(train_history).plot(figsize=(8, 5), title=model_key)
    plt.grid(True)
    # plt.gca().set_ylim(0, 1)
    # plt.show()
  except:
    pass
```

Plot the Selected Model

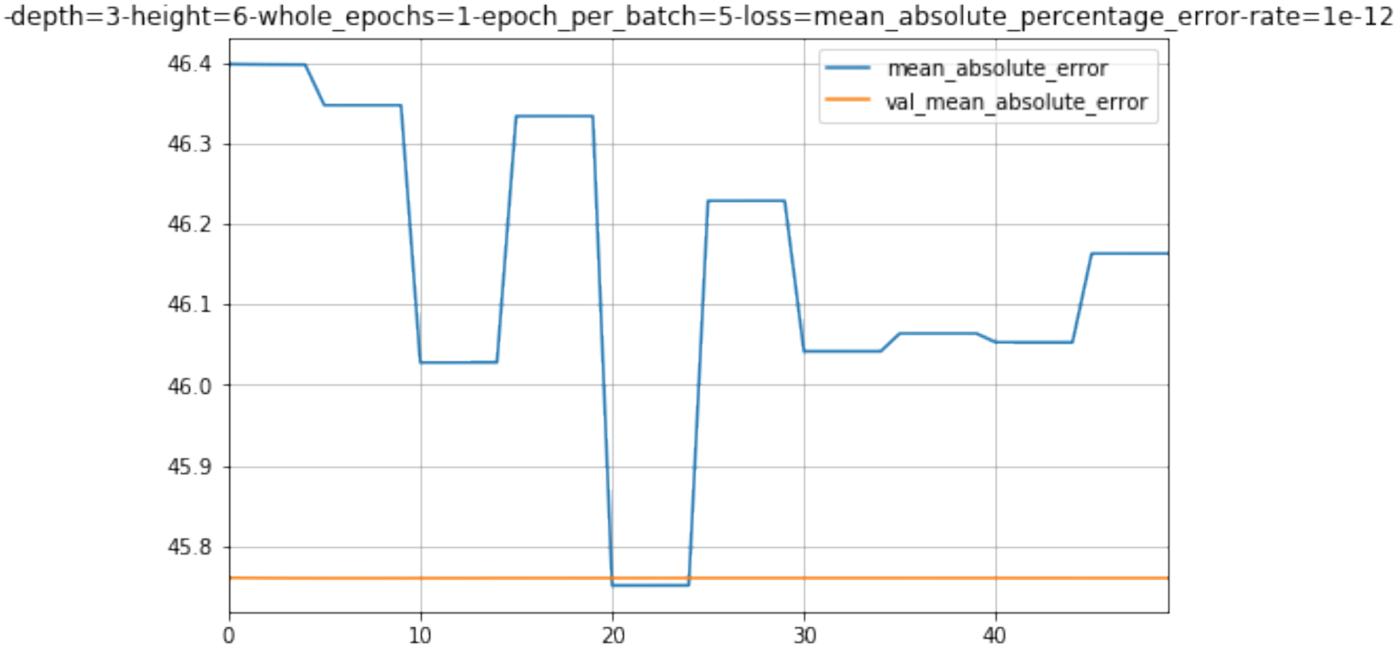
```
plot_learning_curves(chosen_key, model_train_history[chosen_key].loss_history)
```



```
plot_learning_curves(chosen_key,
model_train_history[chosen_key].percentage_error_history)
```

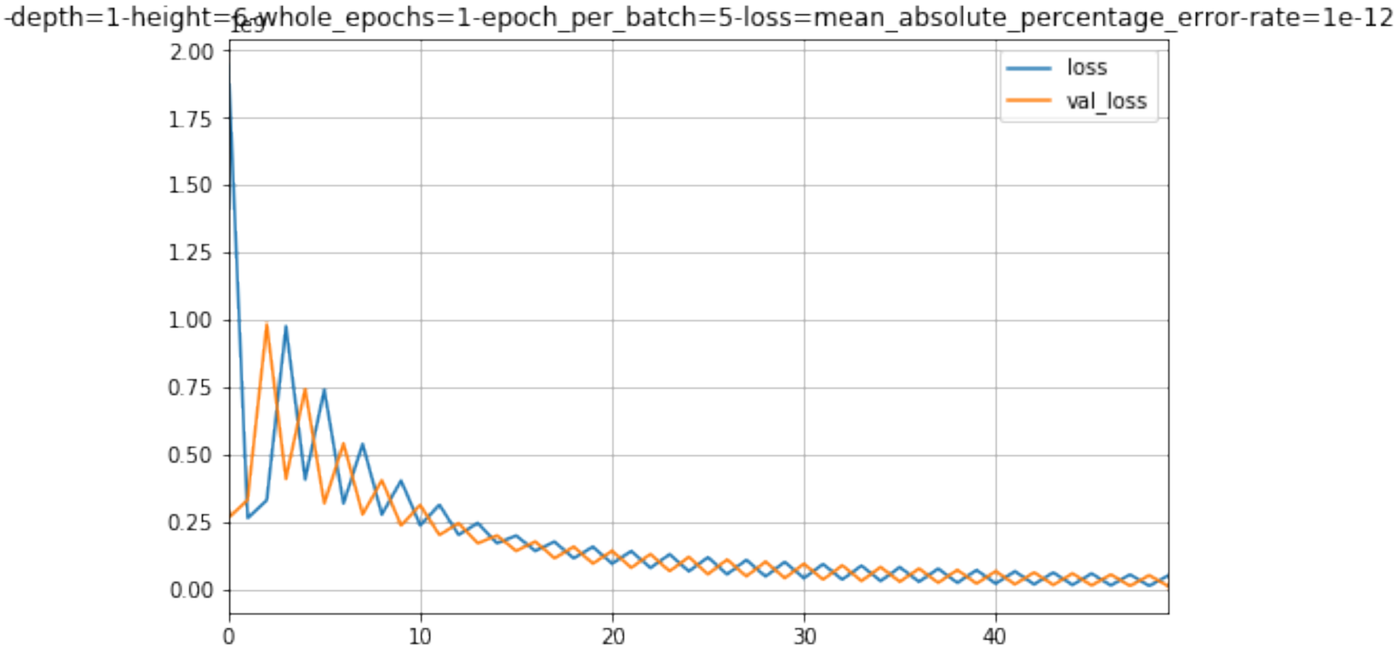


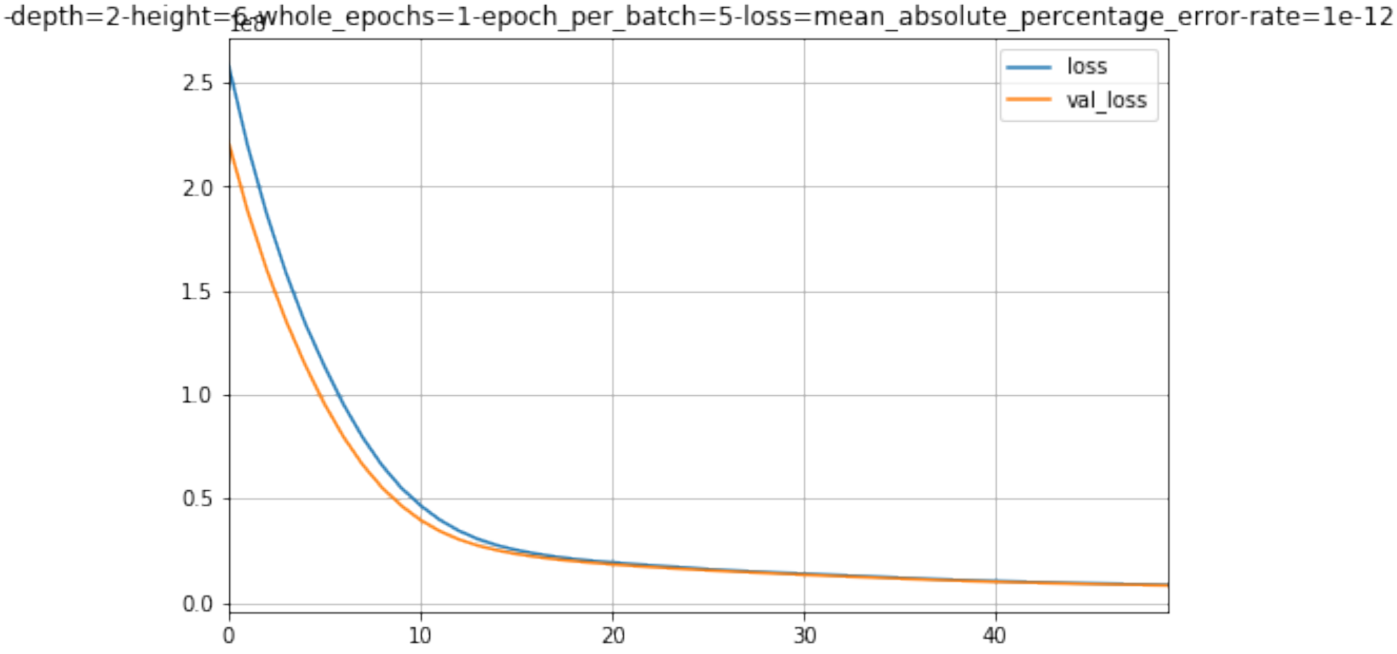
```
plot_learning_curves(chosen_key, model_train_history[chosen_key].error_history)
```

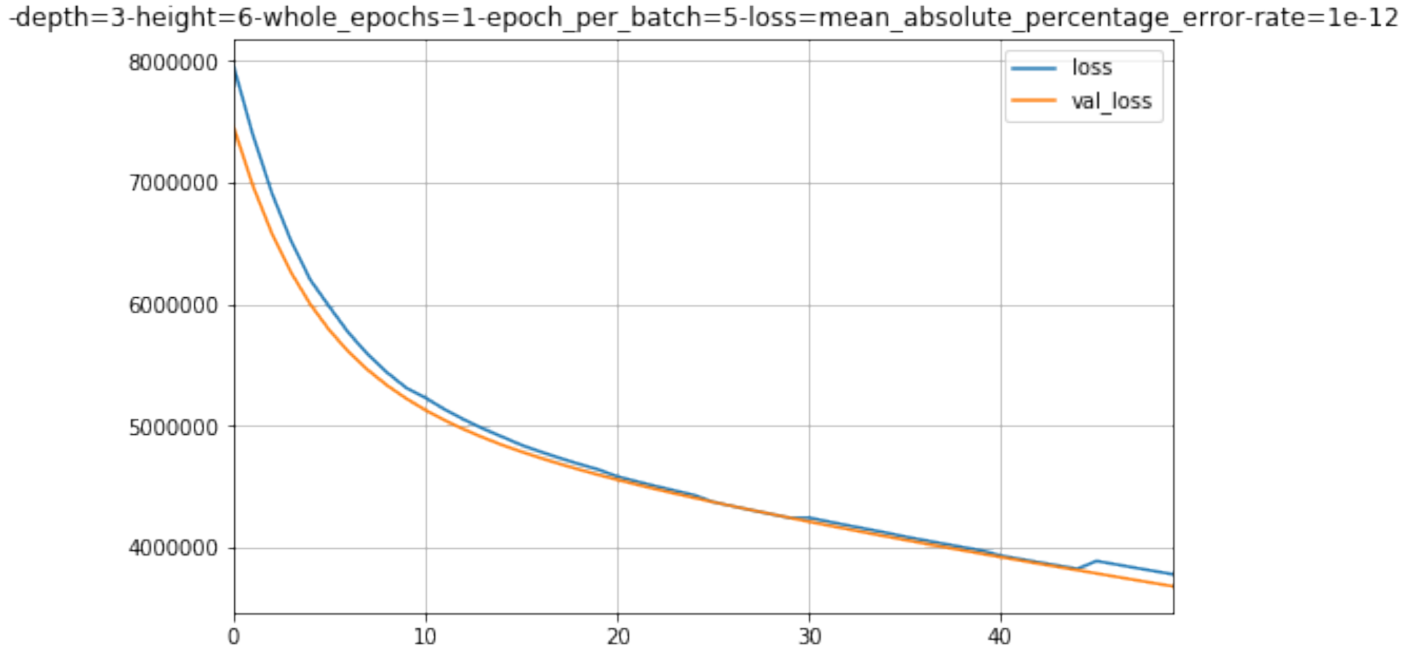


Plot All Models

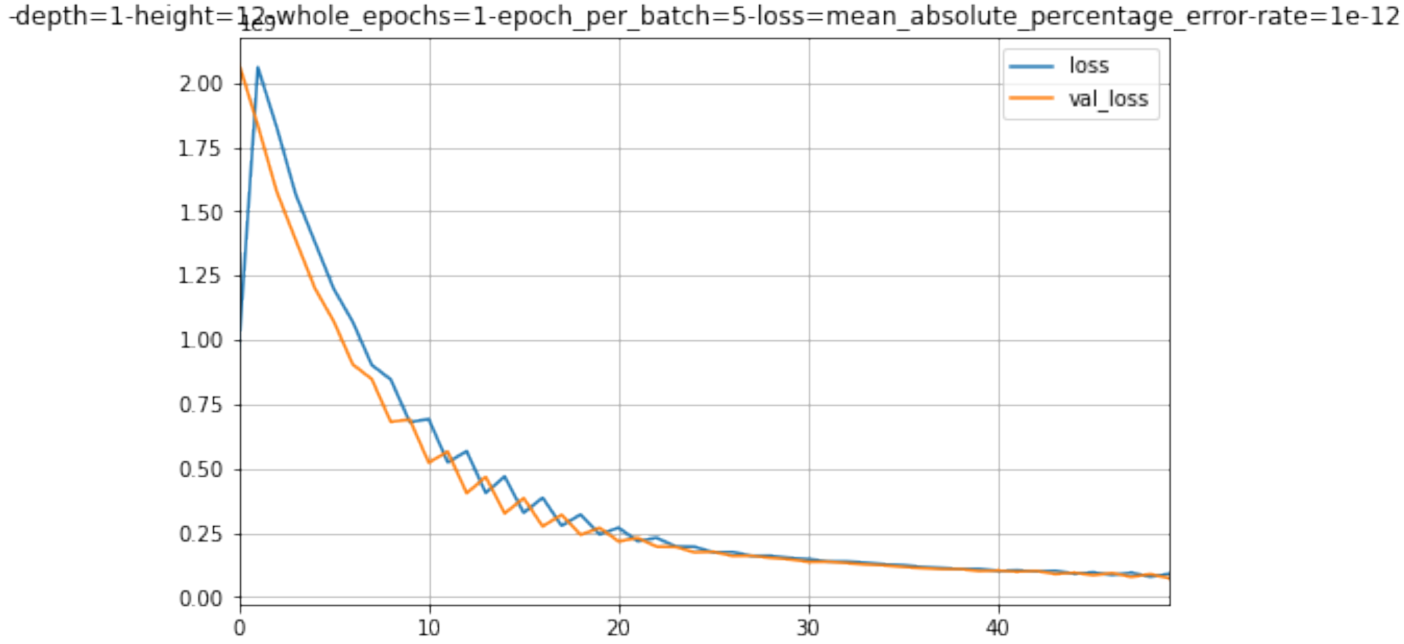
```
for key in model_train_history.keys():
  plot_learning_curves(key, model_train_history[key].loss_history)
```


-depth=1-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12
1e9
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
10
20
30
40
loss
val_loss


-depth=2-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12
1e9
2.5
2.0
1.5
1.0
0.5
0.0
0
10
20
30
40
loss
val_loss

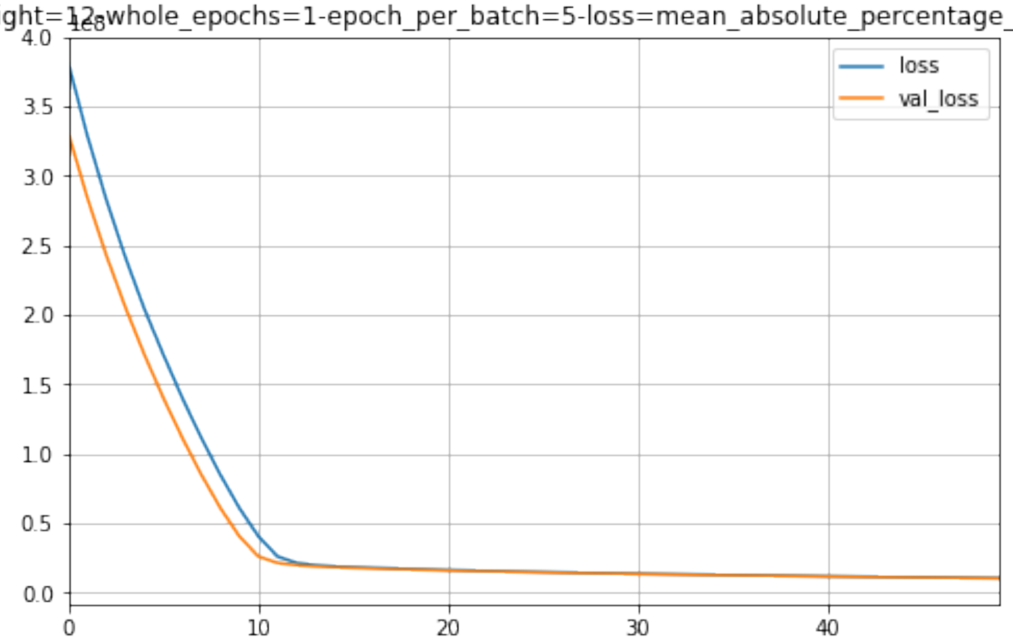


-depth=3-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12

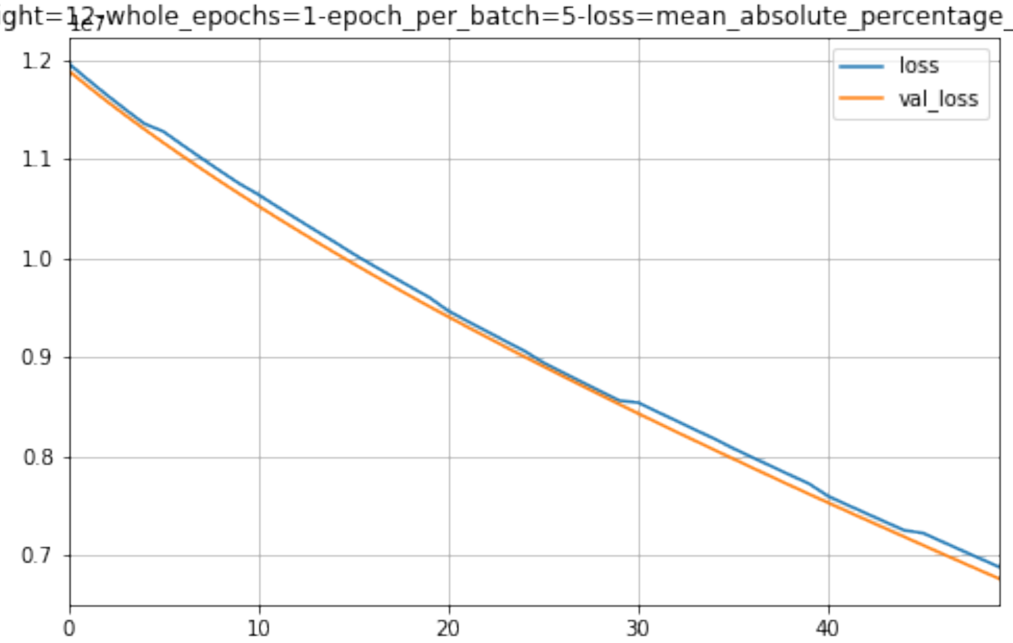


-depth=1-height=12-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12

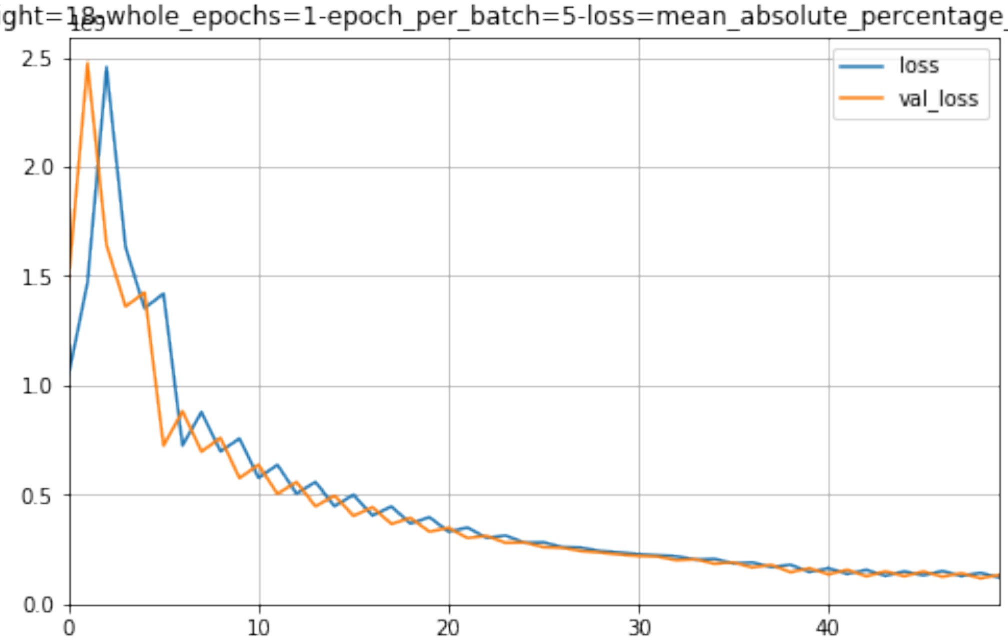
-depth=2-height=12-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12



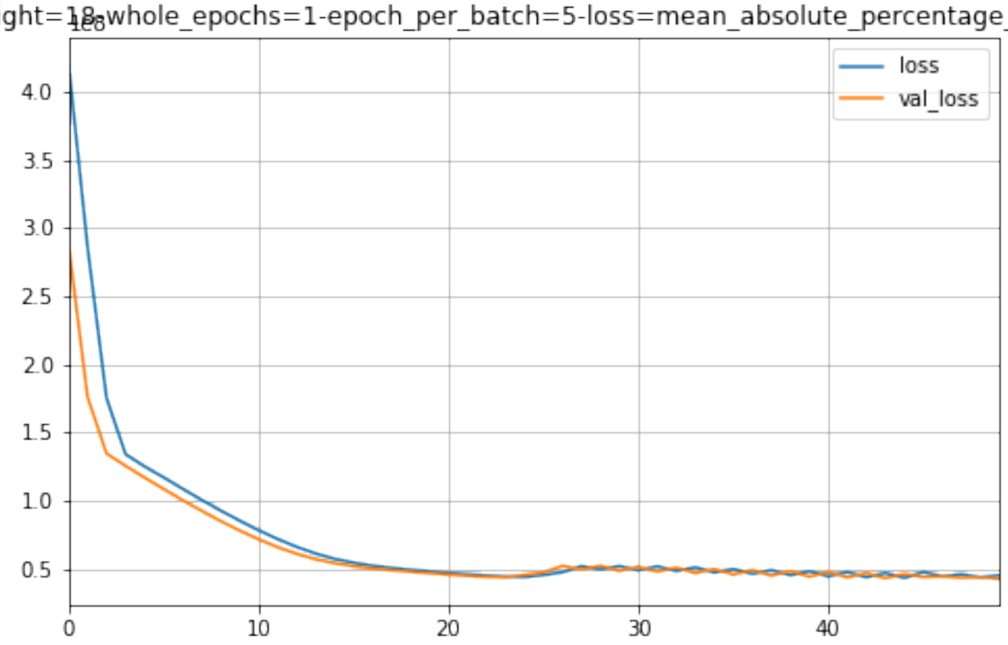
-depth=3-height=12-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12

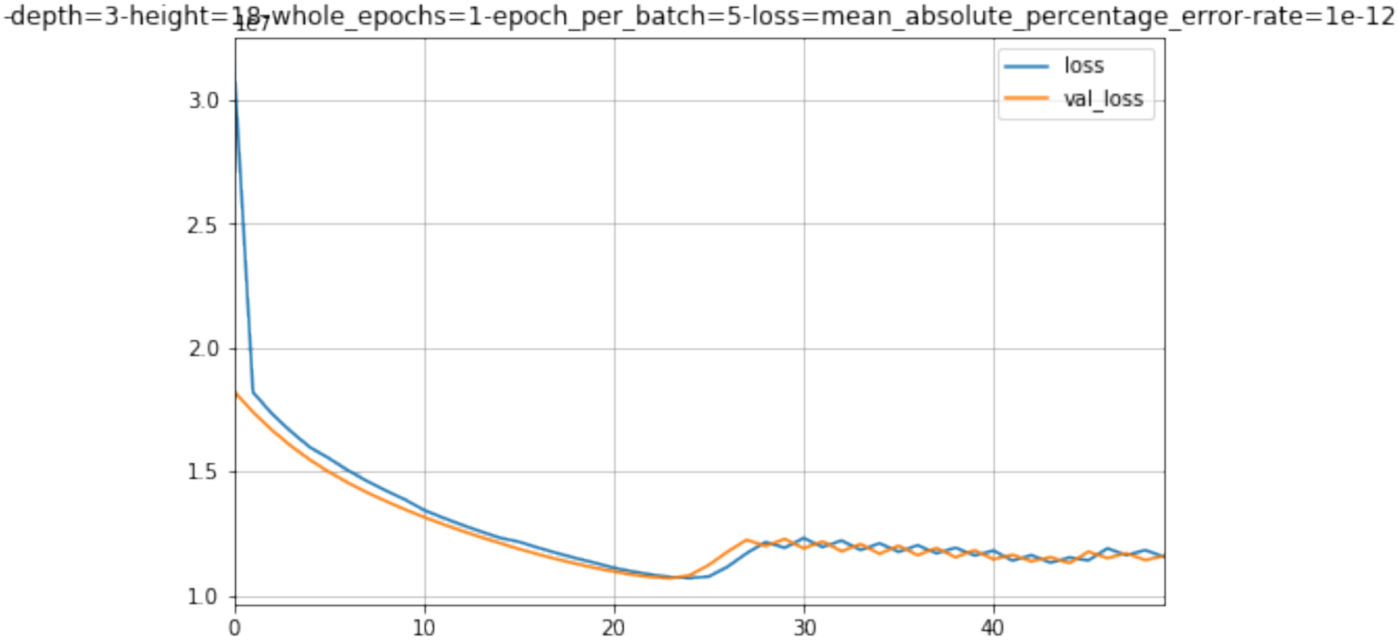


-depth=1-height=18-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12

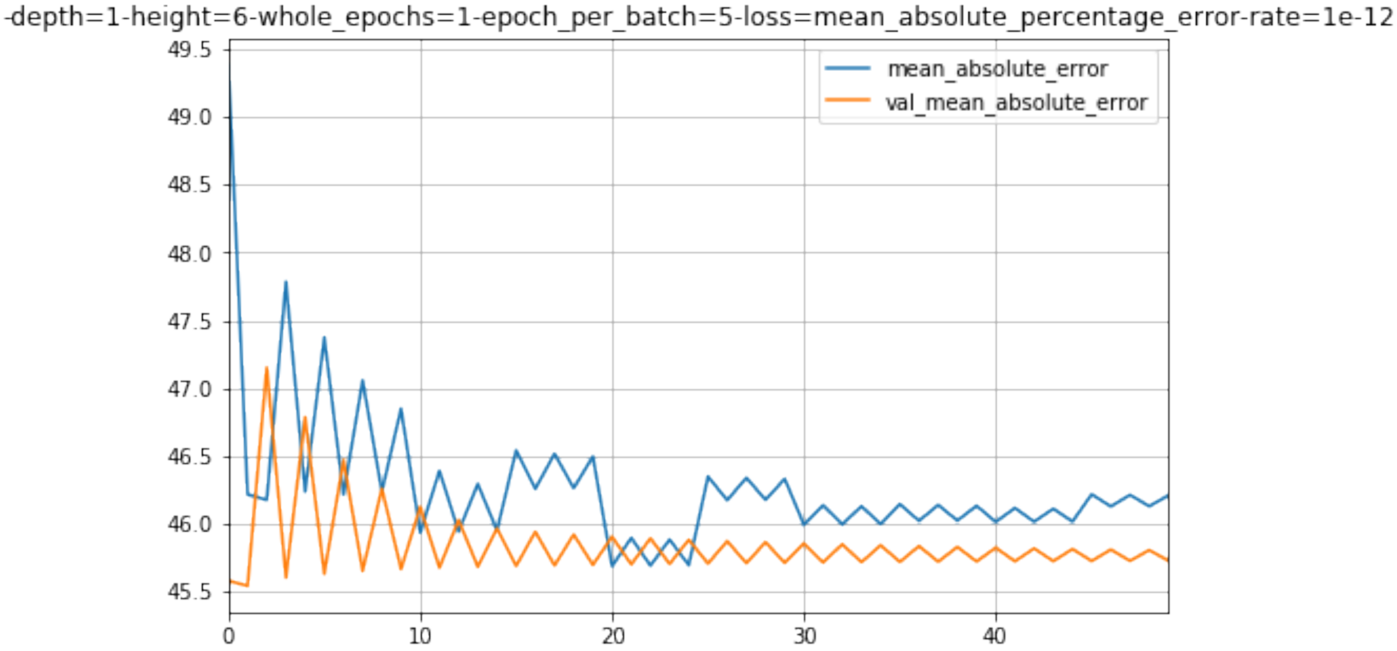


-depth=2-height=18-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12

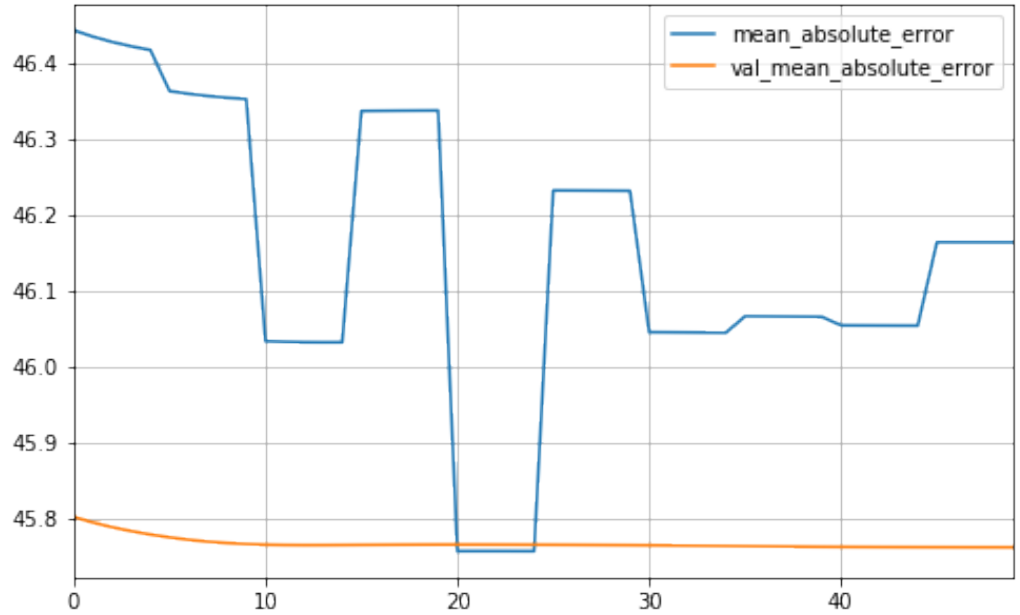




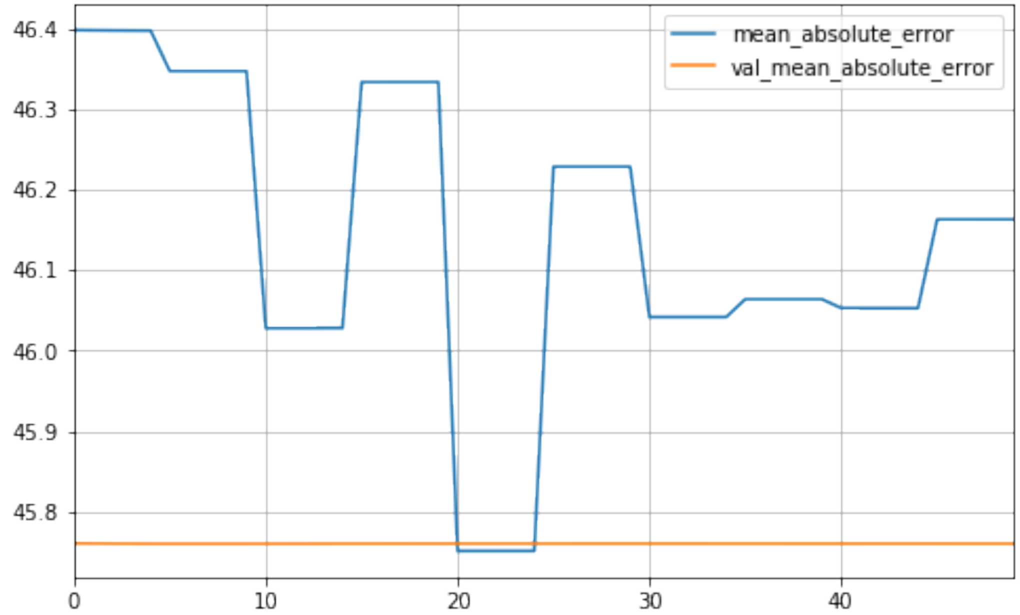
```
for key in model_train_history.keys():
  plot_learning_curves(key, model_train_history[key].error_history)
```



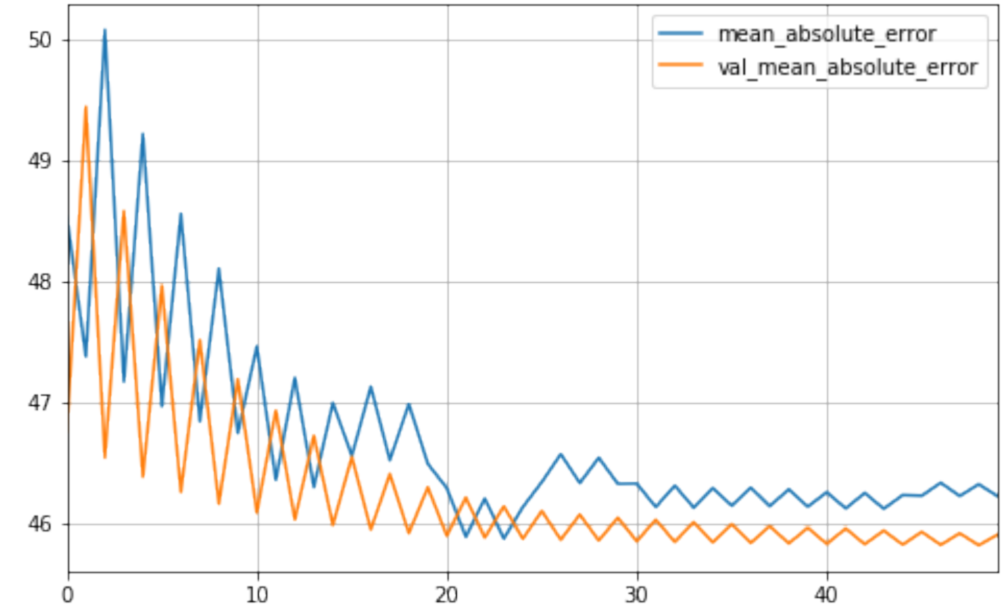
-depth=2-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12



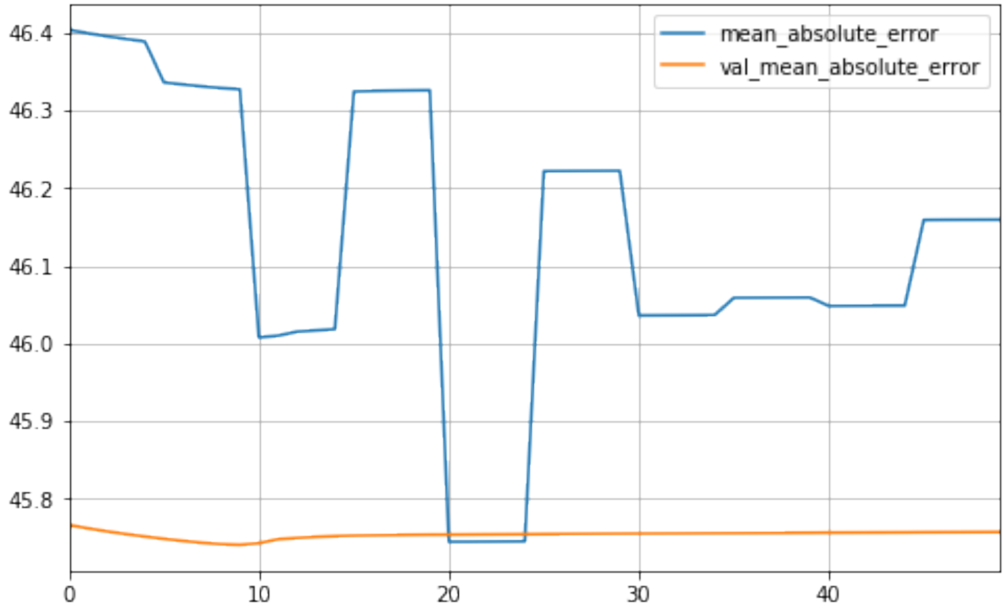
-depth=3-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12

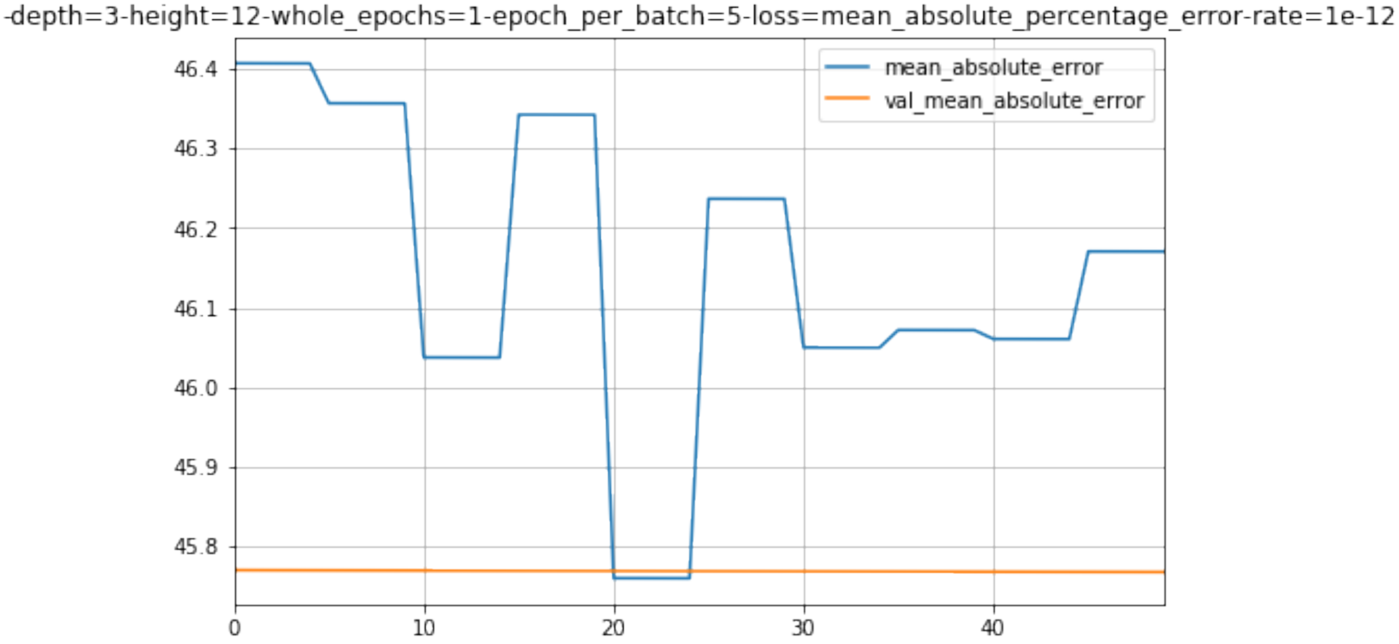


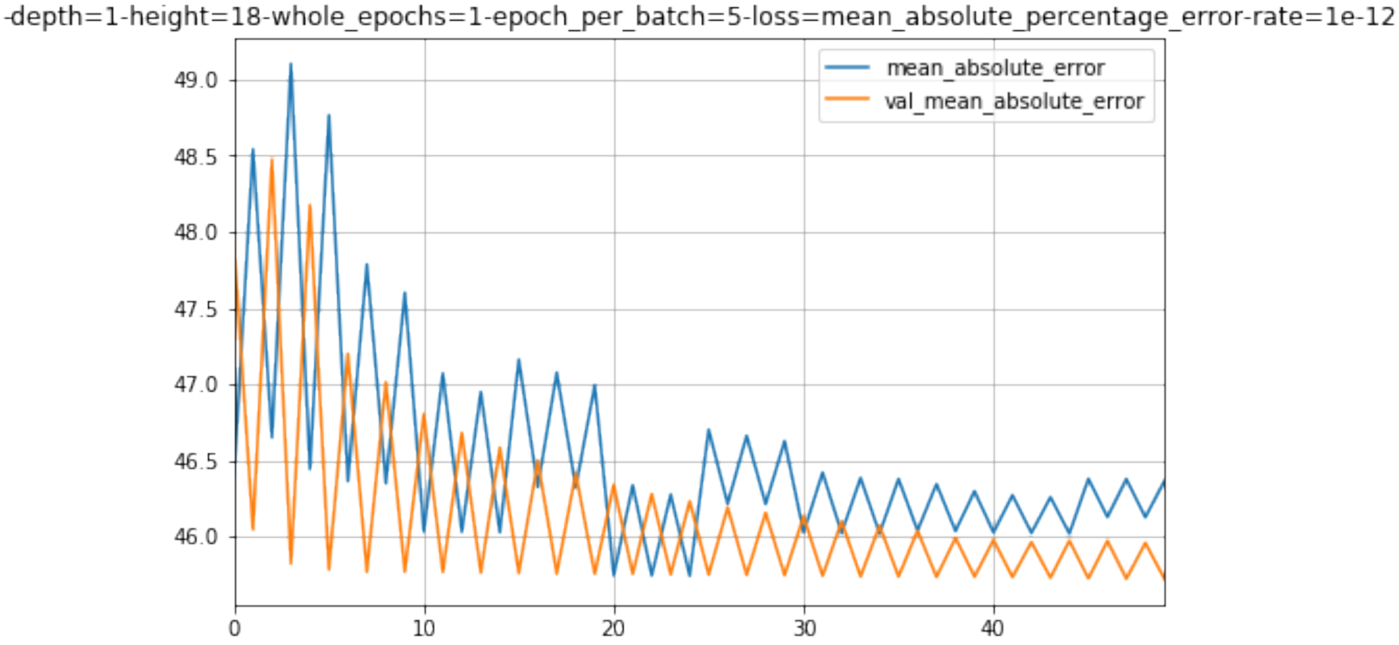
-depth=1-height=12-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12



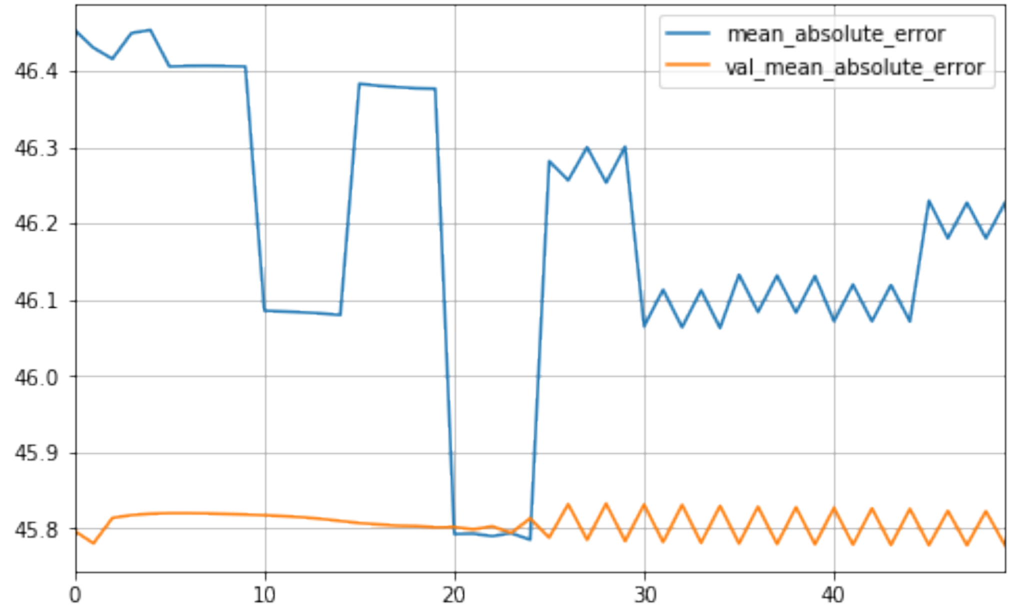
-depth=2-height=12-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12



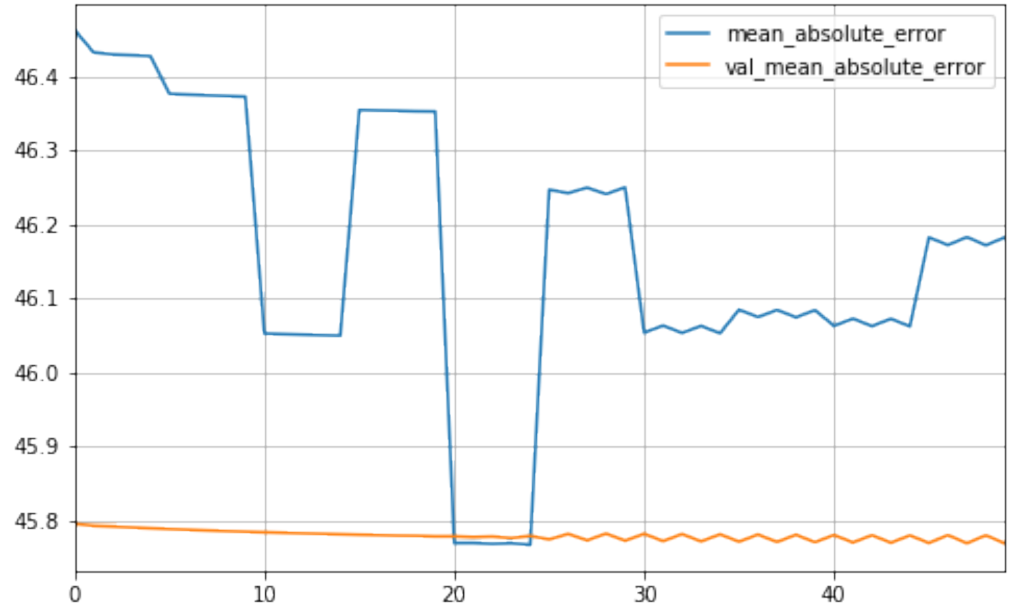

-depth=3-height=12-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12
mean_absolute_error
val_mean_absolute_error
46.4
46.3
46.2
46.1
46.0
45.9
45.8
0
10
20
30
40

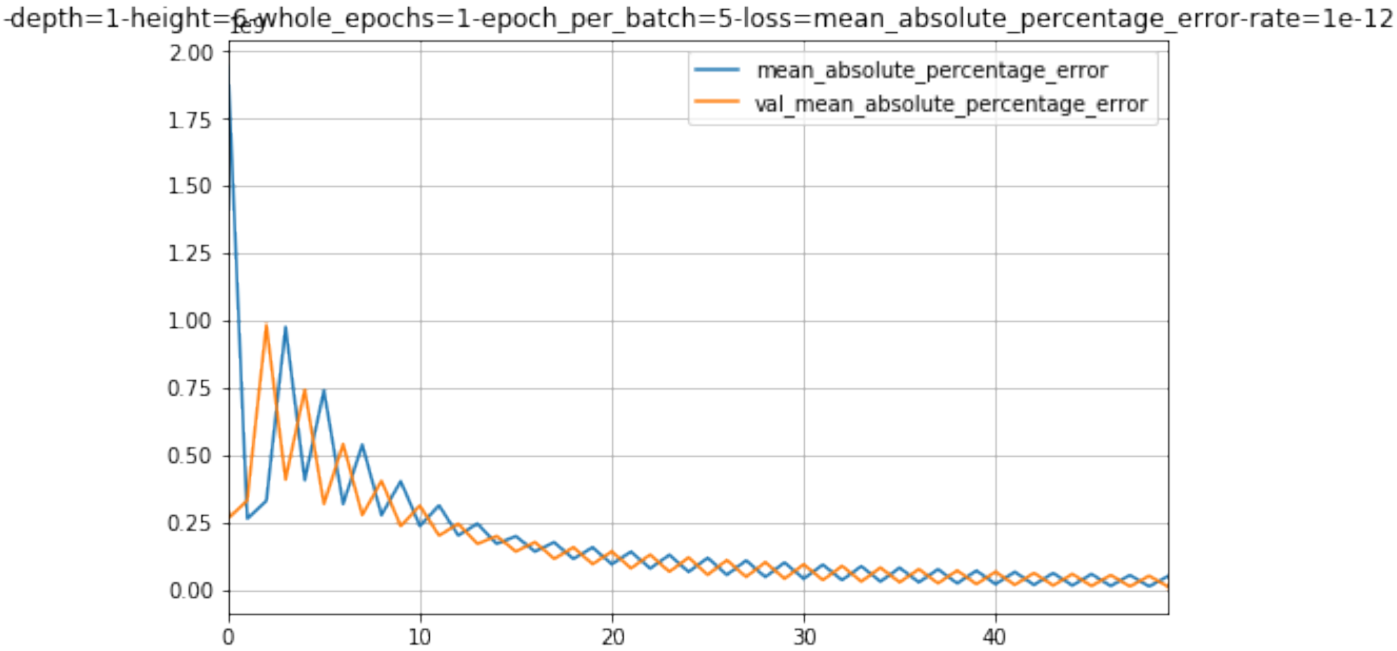

-depth=1-height=18-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12
mean_absolute_error
val_mean_absolute_error
49.0
48.5
48.0
47.5
47.0
46.5
46.0
0
10
20
30
40

-depth=2-height=18-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12

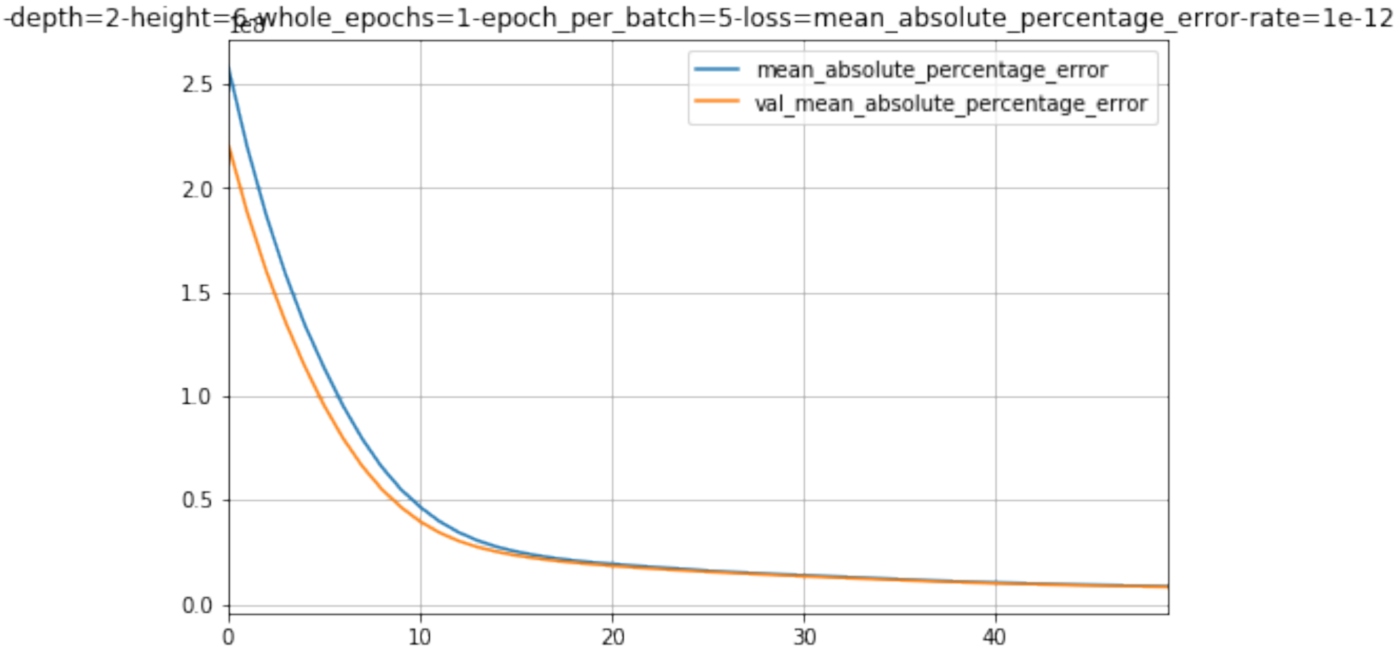


-depth=3-height=18-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12

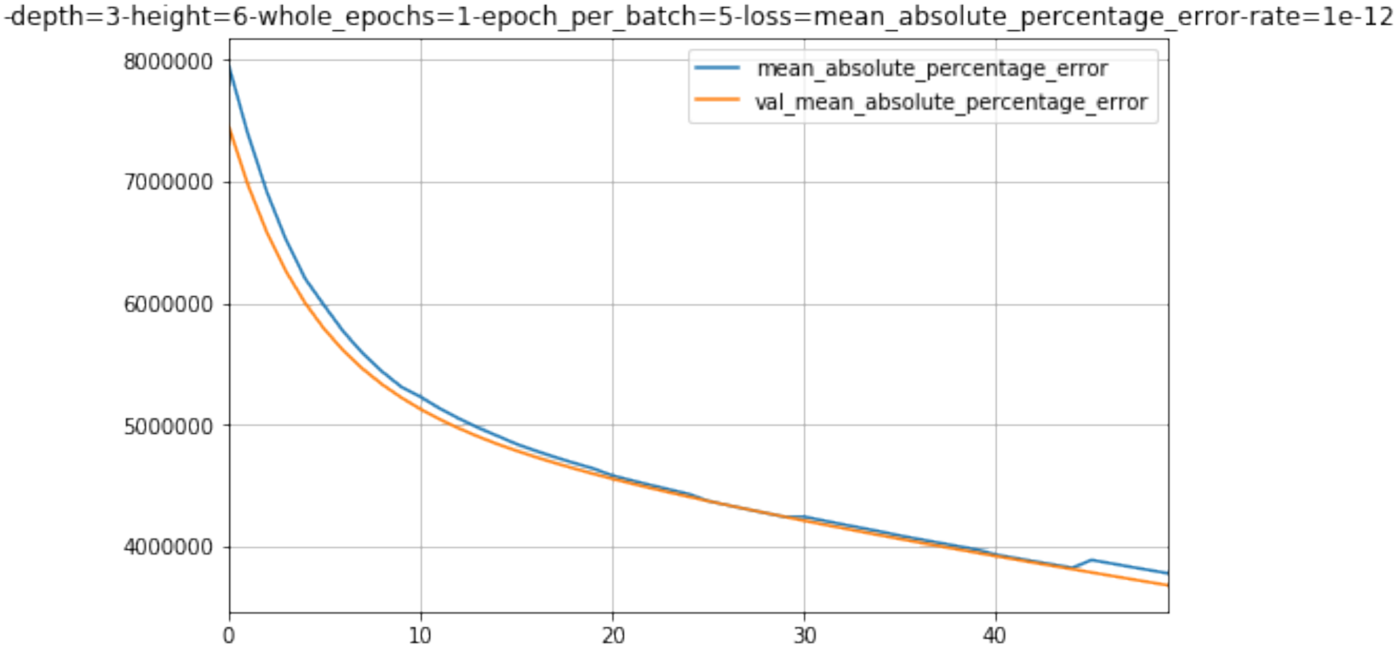


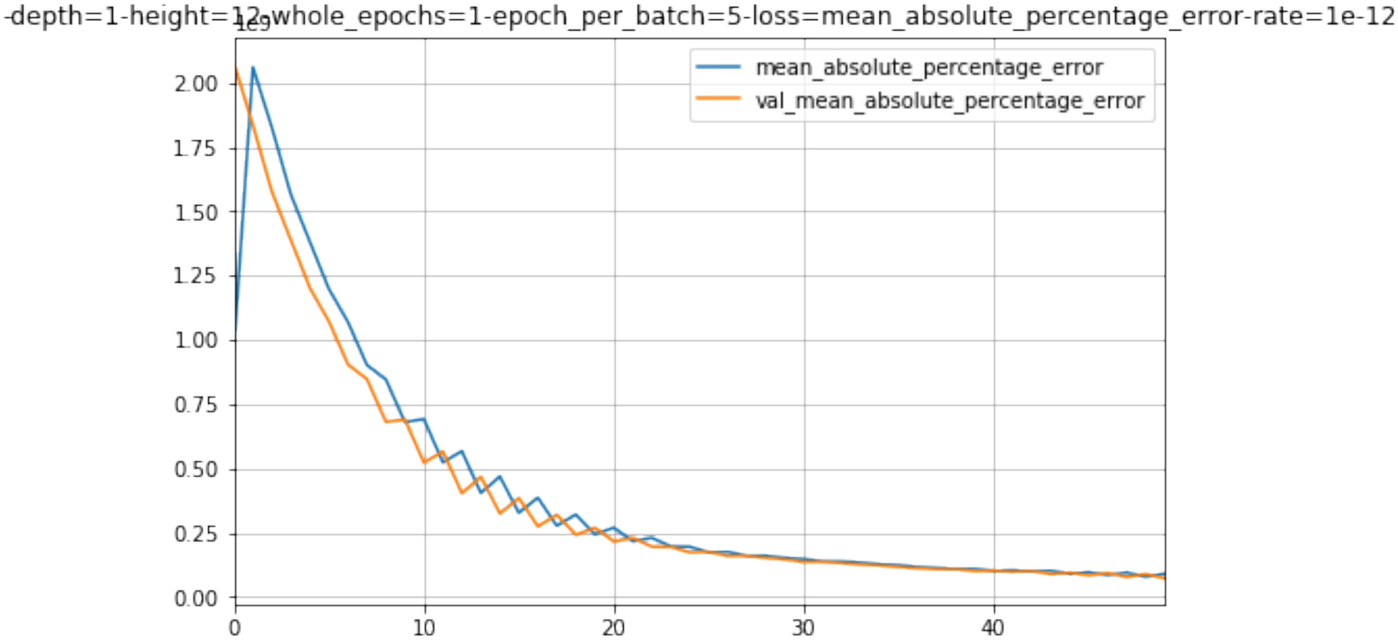


-depth=1-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12

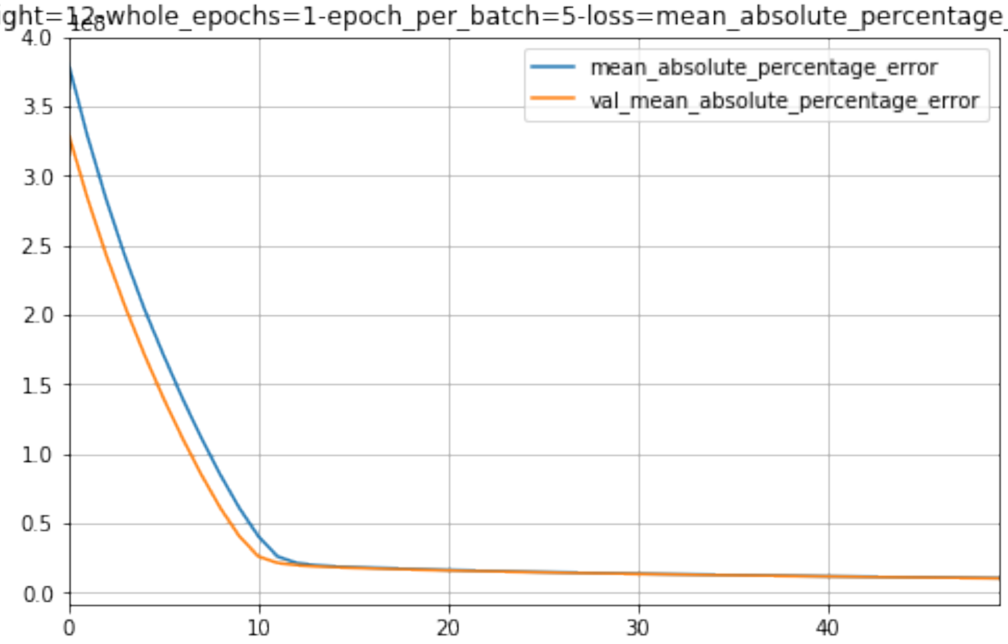


-depth=2-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12

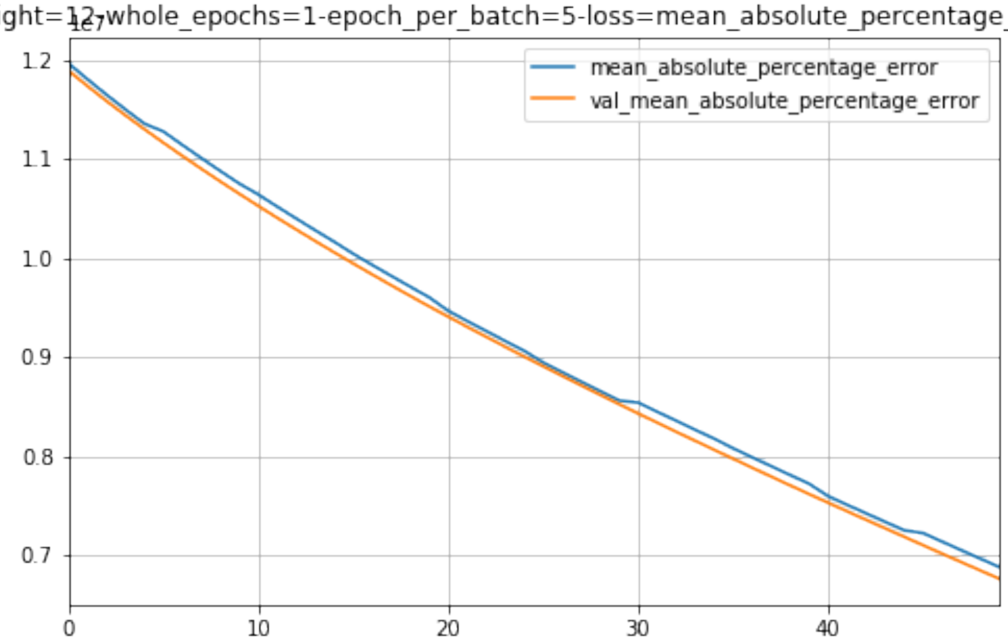

-depth=3-height=6-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12
mean_absolute_percentage_error
val_mean_absolute_percentage_error
8000000
7000000
6000000
5000000
4000000
0
10
20
30
40

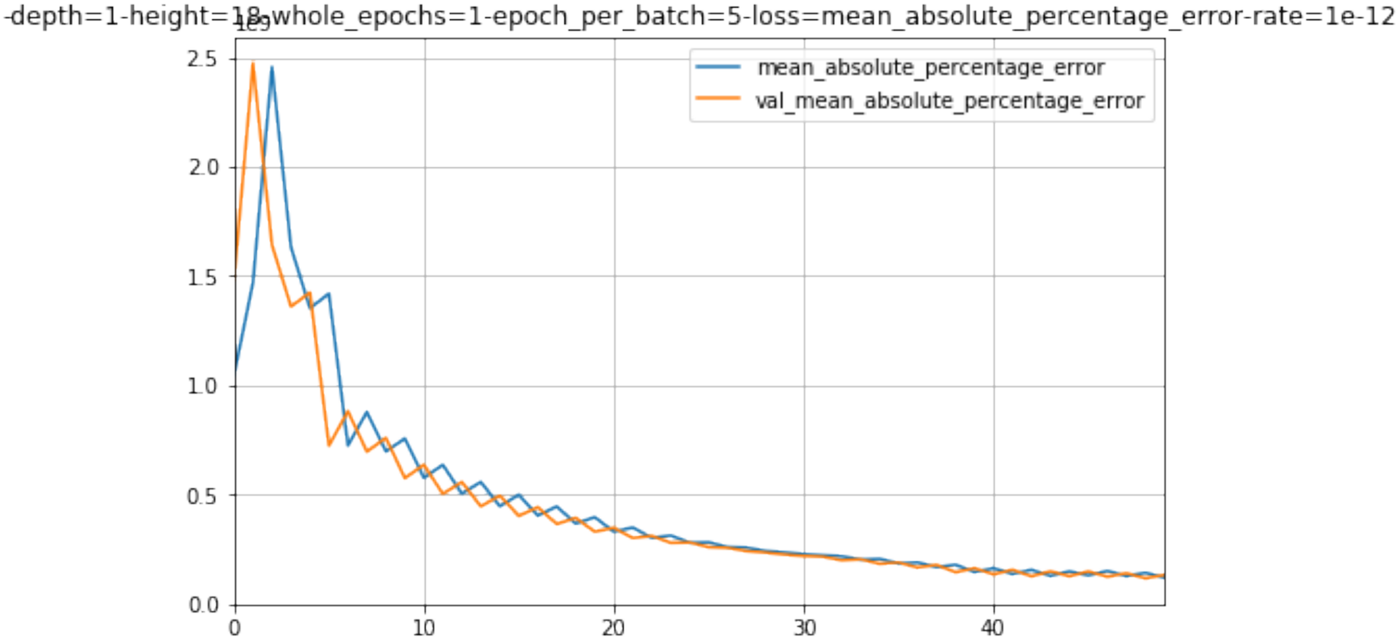

-depth=1-height=12-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12
1e9
mean_absolute_percentage_error
val_mean_absolute_percentage_error
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
10
20
30
40

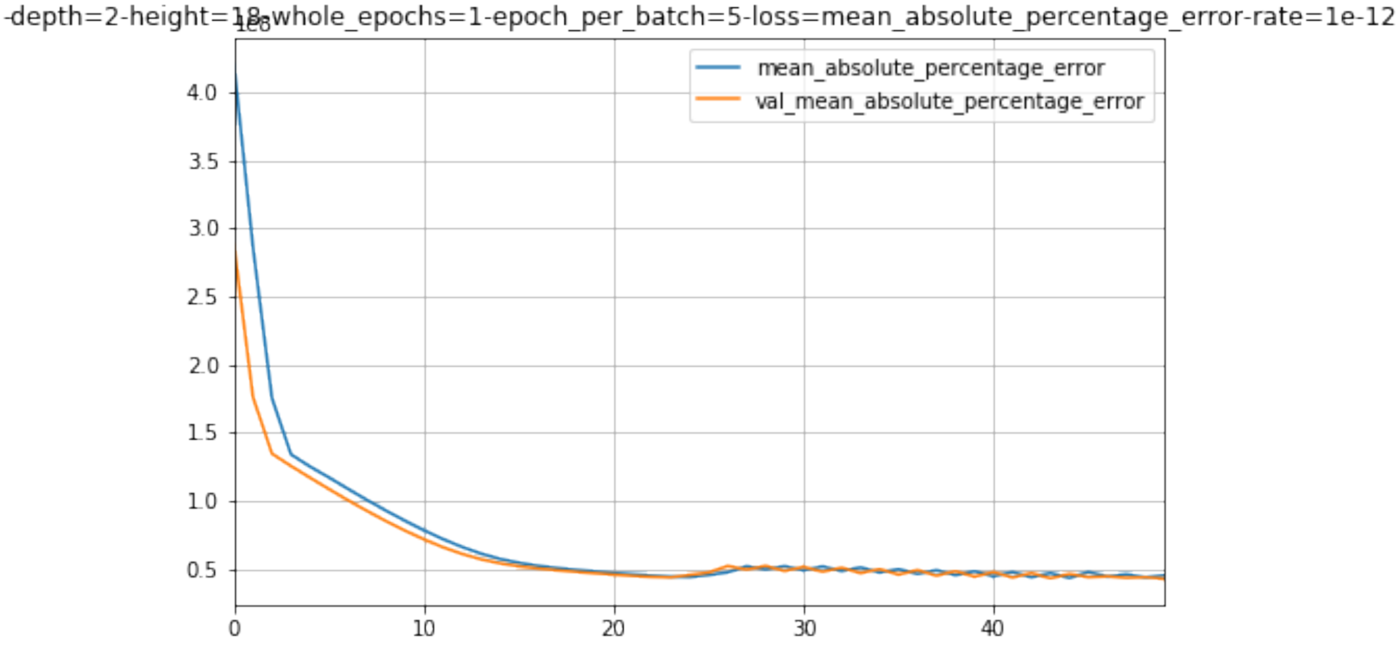
-depth=2-height=12-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12

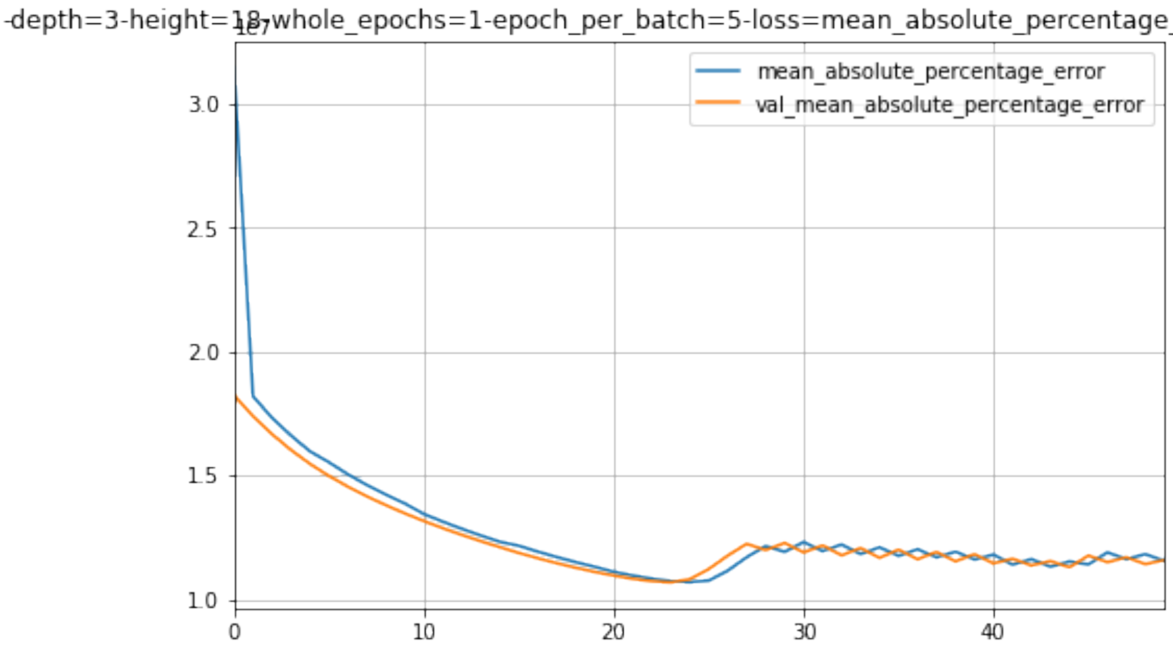


-depth=3-height=12-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12




-depth=1-height=18-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12


-depth=2-height=18-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12



-depth=3-height=18-whole_epochs=1-epoch_per_batch=5-loss=mean_absolute_percentage_error-rate=1e-12