# Train/Validation Split

# Choose the first 83 examples for training.

training\_examples = preprocess\_features(concrete\_dataframe.head(83))

training\_targets = preprocess\_targets(concrete\_dataframe.head(83))

# Choose the 20 examples for validation.

validation\_examples = preprocess\_features(concrete\_dataframe.tail(20))

validation\_targets = preprocess\_targets(concrete\_dataframe.tail(20))

# Double-check that we've done the right thing.

print("Training examples summary:")

display.display(training\_examples.describe())

print("Validation examples summary:")

display.display(validation\_examples.describe())

print("Training targets summary:")

display.display(training\_targets.describe())

print("Validation targets summary:")

display.display(validation\_targets.describe())

# Build Model

baseline\_model = keras.Sequential([

    keras.layers.Dense(4, activation=tf.nn.relu,

                       input\_shape=(training\_examples.shape[1],)),

    keras.layers.Dense(1)

  ])

baseline\_model.compile(loss='mse',

                optimizer="rmsprop",

                metrics=['mae'])

baseline\_model.summary()

**Fit Model**

class PrintDot(keras.callbacks.Callback):

def on\_epoch\_end(self, epoch, logs):

if epoch % 100 == 0: print('')

print('.', end='')

EPOCHS = 500

b\_history = baseline\_model.fit(training\_examples, training\_targets, epochs=EPOCHS,

validation\_data= (validation\_examples, validation\_targets), verbose=0,

callbacks=[PrintDot()])

**Normalized Training and Validation data**

# Choose the first 83 examples for training.

training\_examples = preprocess\_features(concrete\_dataframe.head(83))

training\_targets = preprocess\_targets(concrete\_dataframe.head(83))

scaler = StandardScaler().fit(training\_examples.values)

scaledf = scaler.transform(training\_examples.values)

training\_examples = pd.DataFrame(scaledf, index=training\_examples.index, columns=training\_examples.columns)

# Choose the 20 examples for validation.

validation\_examples = preprocess\_features(concrete\_dataframe.tail(20))

vscaled = scaler.transform(validation\_examples.values)

validation\_examples = pd.DataFrame(vscaled, index=validation\_examples.index, columns=validation\_examples.columns)

validation\_targets = preprocess\_targets(concrete\_dataframe.tail(20))

**Build Model Regularized**

l1\_model = keras.Sequential([

    keras.layers.Dense(32, kernel\_regularizer=keras.regularizers.l1(0.1), activation=tf.nn.relu,

                       input\_shape=(training\_examples.shape[1],)),

    keras.layers.Dense(32, use\_bias=True, kernel\_regularizer=keras.regularizers.l1(0.01), activation=tf.nn.relu),

    keras.layers.Dense(32, use\_bias=True, kernel\_regularizer=keras.regularizers.l1(0.01), activation=tf.nn.relu),

    keras.layers.Dense(32, use\_bias=True, kernel\_regularizer=keras.regularizers.l1(0.01), activation=tf.nn.relu),

    keras.layers.Dense(1)

  ])

l1\_model.compile(loss='mse',

                optimizer="rmsprop",

                metrics=['mae'])

l2\_model = keras.Sequential([

    keras.layers.Dense(10, kernel\_regularizer=keras.regularizers.l2(0.1), activation=tf.nn.relu,

                       input\_shape=(training\_examples.shape[1],)),

    keras.layers.Dropout(0.25),

    keras.layers.Dense(10, kernel\_regularizer=keras.regularizers.l2(0.1), activation=tf.nn.relu),

    keras.layers.Dropout(0.25),

    keras.layers.Dense(10, kernel\_regularizer=keras.regularizers.l2(0.1), activation=tf.nn.relu),

    keras.layers.Dropout(0.25),

    keras.layers.Dense(10, kernel\_regularizer=keras.regularizers.l2(0.1), activation=tf.nn.relu),

    keras.layers.Dense(1)

  ])

l2\_model.compile(loss='mse',

                optimizer="rmsprop",

                metrics=['mae'])

# Plot history Multiple

plot\_history([('baseline', b\_history),

('L1', l1\_history),

('L2', l2\_history)])

**Prediction**

#Prediction

valpreds = baseline\_model.predict\_on\_batch(validation\_examples)

print(valpreds)

#Targets

with pd.option\_context('display.max\_rows', None, 'display.max\_columns', None):

print(validation\_targets)

# Plot Results

import matplotlib.pyplot as plt

def plot\_history(histories, key='mae'):

plt.figure(figsize=(16,10))

for name, history in histories:

val = plt.plot(history.epoch, history.history['val\_'+key],

'--', label=name.title()+' Val')

plt.plot(history.epoch, history.history[key], color=val[0].get\_color(),

label=name.title()+' Train')

plt.xlabel('Epochs')

plt.ylabel(key.replace('\_',' ').title())

plt.legend()

plt.xlim([0,max(history.epoch)])

plt.ylim([0,50])

plot\_history([('baseline', b\_history)])

**Plot Multiple Model Results**

plot\_history([('baseline', b\_history),

('L1', l1\_history),

('L2', l2\_history)])