In [1]:

```
#import libraries
import pandas as pd
import numpy as np
from tensorflow import keras
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from tensorflow.keras import backend as K
```

In [2]:

```
#import CO2 data and formatting it to numpy array
co2_dir = r"C:\Users\Muji\Documents\Columbia\Courses\Earth and Environmental Engineerin
co2_array = np.array(pd.read_csv(co2_dir))

co2_avg = []
for i in range(2304):
    co2_avg.append(float(co2_array[i]))
time_len = len(co2_avg)
```

In [3]:

```
# Display training progress by printing a single dot for each completed epoch
class PrintDot(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        if epoch % 100 == 0: print('')
        print('.', end='')

# Function to plot how the model is doing during training
# Visualize the model's training progress using the stats stored in the history object.
# We want to use this data to determine how long to train before the model stops making

def plot_history_accuracy(history):
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.plot(history.epoch, np.array(history.history['accuracy']),
           label='Train accuracy')
    plt.plot(history.epoch, np.array(history.history['val_accuracy']),
           label = 'Val accuracy')
    plt.legend()

def plot_history_mae(history):
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Mean Abs Error')
    plt.plot(history.epoch, np.array(history.history['mae']),
           label='Train Loss')
    plt.plot(history.epoch, np.array(history.history['val_mae']),
           label = 'Val loss')
    plt.legend()
```

Date-to-date Regression

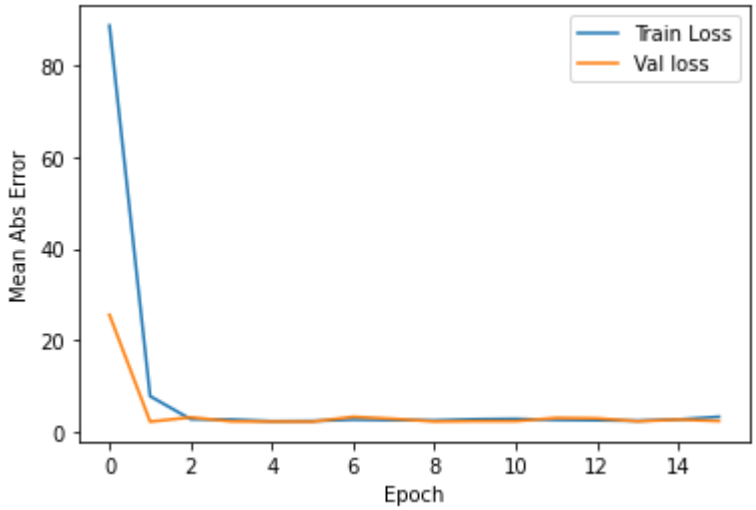
In [22]:
```
#creates training and testing data using only the previous date to predict the next
x = np.array(co2_avg[:-1])
y = np.array(co2_avg[1:])

X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size = 0.2)
```

In [23]:
```
#creating the simple neural network
model_nn_single = keras.Sequential([
    keras.layers.Dense(5, input_shape=((1,)), activation=tf.nn.relu),
    keras.layers.Dense(5, activation=tf.nn.relu),
    #keras.layers.Dropout(0.1),
    keras.layers.Dense(1)])

model_nn_single.compile(loss='mse',optimizer='adam',metrics=['mae'])

model_nn_single.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 5)                 10
_________________________________________________________________
dense_5 (Dense)              (None, 5)                 30
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 6
=================================================================
Total params: 46
Trainable params: 46
Non-trainable params: 0
_________________________________________________________________
```

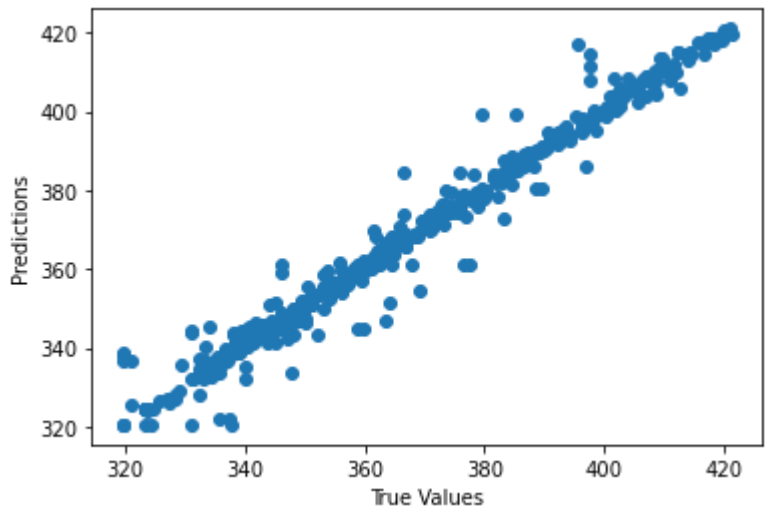
In [24]:
```
#training data + early stopping to avoid overfitting

early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)

# Store training stats
K.set_value(model_nn_single.optimizer.learning_rate, 0.07)
history = model_nn_single.fit(X_train, Y_train, epochs=200,
                    validation_split=0.2, verbose=0,
                    callbacks=[early_stop, PrintDot()])

plot_history_mae(history)

#calculate the final mean average error
[loss, mae] = model_nn_single.evaluate(X_test, Y_test, verbose=0)
print("Testing set Mean Abs Error: {}".format(mae ))
```

```
..............Testing set Mean Abs Error: 2.552804470062256
```



In [25]:
```
#plot the correspondence between predictions and test labels
test_predictions = model_nn_single.predict(X_test)
test_labels = Y_test

plt.scatter(Y_test, test_predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.plot()
```

Out[25]: []



Multi-date Regression

In [25]:
```
length = 4

X = []
Y = []
for i in range(time_len-length):
    features = []
    for j in range(length):
        features.append(co2_avg[i+j])
    X.append(np.array(features))
    Y.append(np.array([co2_avg[i+length]]))
```

```
x = np.array(X)
y = np.array(Y)

X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size = 0.2)
```

In [32]:

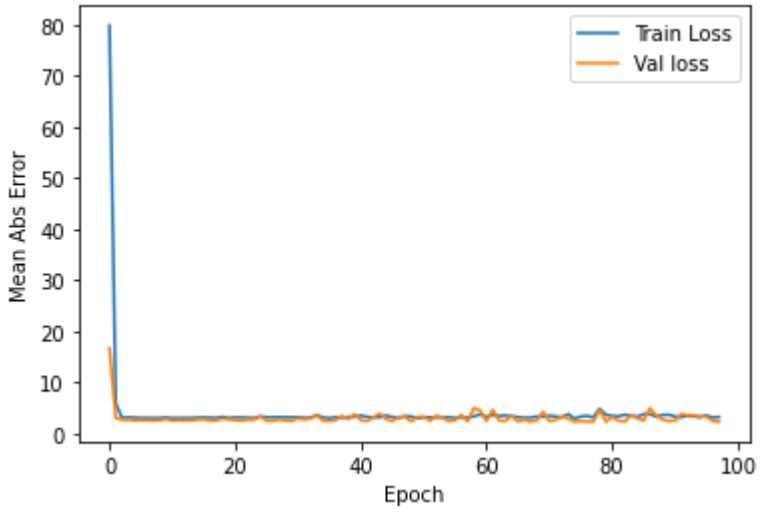
```
model_nn_multiple = keras.Sequential([
    keras.layers.Dense(15, input_shape=((length,)), activation=tf.nn.relu),
    keras.layers.Dense(15, activation=tf.nn.relu),
    #keras.layers.Dropout(0.2),
    keras.layers.Dense(1)])

model_nn_multiple.compile(loss='mse',optimizer='adam',metrics=['mae'])

model_nn_multiple.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_21 (Dense)             (None, 15)                75
_________________________________________________________________
dense_22 (Dense)             (None, 15)                240
_________________________________________________________________
dense_23 (Dense)             (None, 1)                 16
=================================================================
Total params: 331
Trainable params: 331
Non-trainable params: 0
_________________________________________________________________
```

In [33]:

```
# If you train too long, you are prone to over-fitting
# this prevents the model from generalizing to data it has never seen before
# early stopping is one way to go about this
# The patience parameter is the amount of epochs to check for improvement
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=15)

K.set_value(model_nn_multiple.optimizer.learning_rate, 0.01)

# Store training stats
history = model_nn_multiple.fit(X_train, Y_train, epochs=200,
                    validation_split=0.2, verbose=0,
                    callbacks=[early_stop, PrintDot()])

plot_history_mae(history)

[loss, mae] = model_nn_multiple.evaluate(X_test, Y_test, verbose=0)
print("Testing set Mean Abs Error: {}".format(mae ))
```

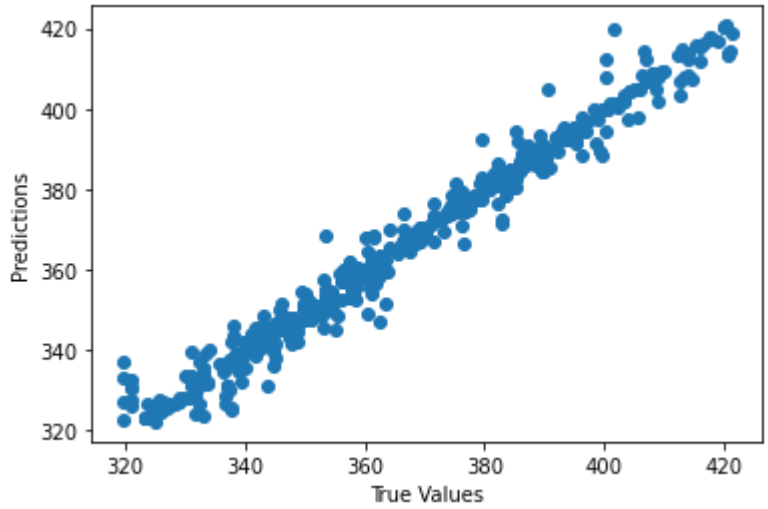
```
.....................................................................................
Testing set Mean Abs Error: 2.698936939239502
```



In [34]:

```
test_predictions = model_nn_multiple.predict(X_test)
test_labels = Y_test

plt.scatter(Y_test, test_predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.plot()
```

Out[34]: []



LSTM

In [61]:

```
#create batches of size length-1
X = []
Y = []
length = 48
for i in range(0,time_len,length):
    batch = []
    for j in range(length-1):
            batch.append(np.array([co2_avg[i+j]]))
    X.append(batch)
    Y.append(np.array([co2_avg[i+length-1]]))
    #Y.append(np.array([temp_avg[i+length-1]]))
```

```
x = np.array(X)
y = np.array(Y)
X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size = 0.2)
```

In [64]:

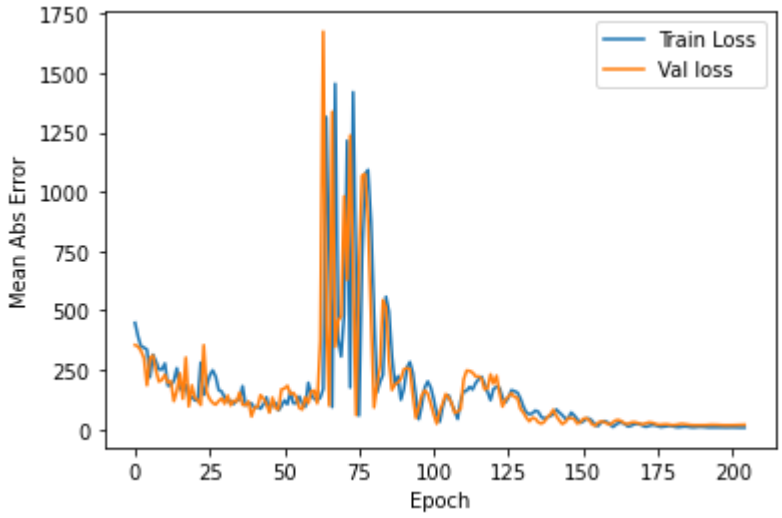
```
model_LSTM = keras.Sequential()
model_LSTM.add(LSTM(150,input_shape=(length-1,1), activation = tf.nn.relu))
model_LSTM.add(Dense(150,activation = tf.nn.relu))
model_LSTM.add(Dense(1))

model_LSTM.compile(loss='mse',optimizer='adam',metrics=['mae'])

model_LSTM.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_11 (LSTM)               (None, 150)               91200
_________________________________________________________________
dense_18 (Dense)             (None, 150)               22650
_________________________________________________________________
dense_19 (Dense)             (None, 1)                 151
=================================================================
Total params: 114,001
Trainable params: 114,001
Non-trainable params: 0
_________________________________________________________________
```

In [65]:

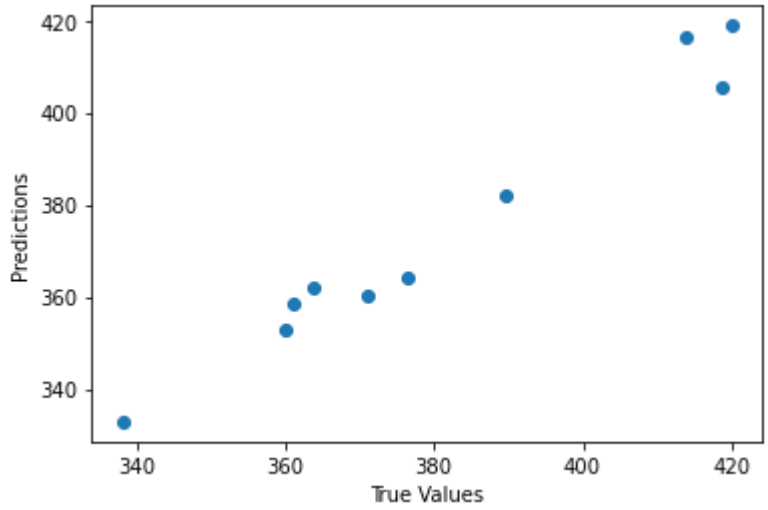
```
# If you train too long, you are prone to over-fitting
# this prevents the model from generalizing to data it has never seen before
# early stopping is one way to go about this
# The patience parameter is the amount of epochs to check for improvement
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=50)

K.set_value(model_LSTM.optimizer.learning_rate, 0.001)

# Store training stats
history = model_LSTM.fit(X_train, Y_train, epochs=1000,
                    validation_split=0.2, verbose=0,
                    callbacks=[early_stop, PrintDot()])

plot_history_mae(history)

[loss, mae] = model_LSTM.evaluate(X_test, Y_test, verbose=0)
print("Testing set Mean Abs Error: {}".format(mae ))
```

```
..........................................................................................

..........................................................................................

.....Testing set Mean Abs Error: 7.3814697265625
```



In [47]:
```
test_predictions = model_LSTM.predict(X_test).flatten()
test_labels = Y_test.flatten()

plt.scatter(test_labels, test_predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.plot()
```

Out[47]: []



As expected, the results are much better using LSTM.

CO2 Predictions

Let's try now to predict future CO2 levels, and create multiple scenarios to use later for our climate prediction data.

In [32]:
```
#create a prediction list with all the predicted values of the model and then use these
length = 2
year = 10
current_values = co2_avg[-length+1:]
prediction = []
```

```
for i in range(year*12*2*2):
    cv = np.array(current_values)
    pred = model_nn_single.predict(cv)
    pred = float(pred)
    current_values.append(pred)
    current_values.pop(0)
    prediction.append(pred)
```

In [48]:

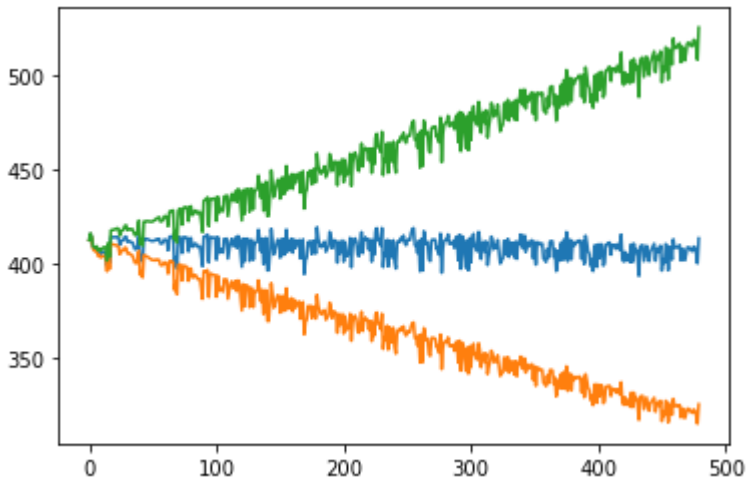
```
#create a prediction list with all the predicted values of the model and then use these
year = 10
current_values = co2_avg[-length+1:]
prediction = []
for i in range(year*12*2*2):
    cv = np.array([[[current_values[i]] for i in range(length-1)]])
    pred = model_LSTM.predict(cv)
    pred = float(pred)
    current_values.append(pred)
    current_values.pop(0)
    prediction.append(pred)
```

In [54]:

```
#create different scenarios with a sinusoidal white noise
future_len = len(prediction)

sinusoidal = np.zeros(future_len)
#sinusoidal = 20*np.sin(np.array([0.15*i for i in range(future_len)]))
normal = prediction[:] + sinusoidal
exp_decreasing = np.exp(-0.0005*np.array([i for i in range(future_len)]))*prediction[:]
exp_increasing = np.exp(0.0005*np.array([i for i in range(future_len)]))*prediction[:]

predictions = [normal,exp_decreasing,exp_increasing]

plt.plot(normal)
plt.plot(exp_decreasing)
plt.plot(exp_increasing)
```

Out[54]:

```
[<matplotlib.lines.Line2D at 0x1a356d067c8>]
```



In [52]:

```
pred_len = len(predictions)
for i in range(pred_len):
    scenario = predictions[i]
```

```
    df = pd.DataFrame(scenario)
    df.to_csv(r"C:\Users\Muji\Documents\Columbia\Courses\Earth and Environmental Engine
```

In [ ]: