[1]:
```
from matplotlib import pyplot as plt
%matplotlib inline

import pandas as pd
import torch
import torch.nn as nn
```

[2]:
```
# URL for monthly mean data
url = 'ftp://aftp.cmdl.noaa.gov/products/trends/co2/co2_mm_mlo.txt'

# Read CSV data from url
data = pd.read_csv(url,
                   delim_whitespace=True,
                   header=52,
                   usecols=[0, 1, 2, 3],
                   names=['Year', 'Month', 'DecimalDate', 'CO2_av'])
display(data)
```

```
     Year  Month  DecimalDate  CO2_av
0    1958      3    1958.2027  315.70
1    1958      4    1958.2877  317.45
2    1958      5    1958.3699  317.51
3    1958      6    1958.4548  317.24
4    1958      7    1958.5370  315.86
..    ...    ...          ...     ...
751  2020     10    2020.7917  411.51
752  2020     11    2020.8750  413.11
753  2020     12    2020.9583  414.25
754  2021      1    2021.0417  415.52
755  2021      2    2021.1250  416.75
```

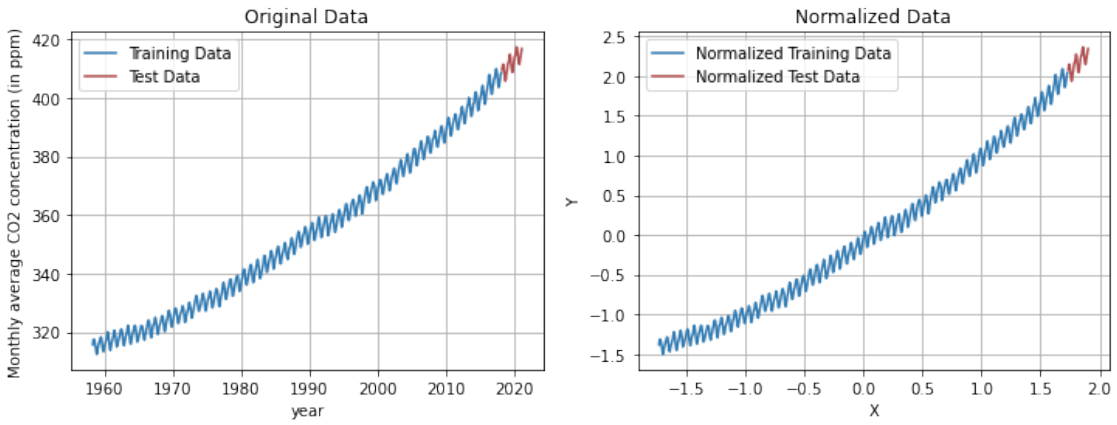
```
[756 rows x 4 columns]
```

[3]:
```
# Convert to pytorch column "vectors"
X = torch.FloatTensor(data['DecimalDate'].values).reshape(-1, 1)
Y = torch.FloatTensor(data['CO2_av']).reshape(-1, 1)

# Use the last TEST_LENGTH months for testing, the rest for training
TEST_LENGTH = 3 * 12

# Training data
X_train = X[:-TEST_LENGTH]
Y_train = Y[:-TEST_LENGTH]

# Test data
X_test  = X[-TEST_LENGTH:]
Y_test  = Y[-TEST_LENGTH:]

# Data for prediction
PRED_LENGTH = 40 * 12
X_pred = (X[-1] + torch.arange(1, PRED_LENGTH + 1) / 12.0).reshape(-1, 1)
```

[4]:
```
def apply_transform(data_list, shift, scale):
    """Return list of normalized data"""

    return [(data - shift) / scale for data in data_list]


# Normalize X data
X_shift = X_train.mean()
X_scale = X_train.std()
X_norm  = apply_transform([X_train, X_test, X_pred],
                          shift=X_shift, scale=X_scale)
[X_train_norm, X_test_norm, X_pred_norm] = X_norm

# Normalize Y data
Y_shift = Y_train.mean()
Y_scale = Y_train.std()
Y_norm  = apply_transform([Y_train, Y_test],
                          shift=Y_shift, scale=Y_scale)
[Y_train_norm, Y_test_norm] = Y_norm
```

[5]:
```
col_train = '#327bb3' # blue
col_test  = '#b04f4f' # red
col_pred  = '#4fb062' # green
```

[6]:
```
# Visualization
fig, axs = plt.subplots(1, 2, figsize=(12, 4))

# Subplot 1: Original data
ax = axs[0]
ax.plot(X_train, Y_train,
        color=col_train, label='Training Data')
ax.plot(X_test,  Y_test,
        color=col_test, label='Test Data')
ax.set_xlabel('year')
ax.set_ylabel('Monthly average CO2 concentration (in ppm)')
ax.set_title('Original Data')
ax.grid()
ax.legend()

# Subplot 2: Normalized data
ax = axs[1]
ax.plot(X_train_norm, Y_train_norm,
        color=col_train, label='Normalized Training Data')
ax.plot(X_test_norm,  Y_test_norm,
        color=col_test,  label='Normalized Test Data')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_title('Normalized Data')
ax.grid()
ax.legend()

plt.show()
```



[7]:
```
# Set seed value for reproducibility
torch.manual_seed(0)

# Compute on GPU, if available
DEVICE = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Choose neural network architecture
model = nn.Sequential(
    nn.Linear(1, 100),
    nn.ReLU(),
    nn.Linear(100, 1)
).to(DEVICE)
print(f'Network nof_parameters = {sum(p.numel() for p in model.parameters())}\n')

# Choose loss function (suitable for regression tasks)
loss_func = torch.nn.MSELoss()

# Choose optimizer and hyperparameters
optimizer = torch.optim.Adam(model.parameters())

# Train for NOF_EPOCHS epochs
NOF_EPOCHS = 500

print('Training Progress...')
for epoch in range(NOF_EPOCHS):

    model.train()

    # Forward pass
    inputs, labels = X_train_norm.to(DEVICE), Y_train_norm.to(DEVICE)
    outputs = model(inputs)
    train_loss = loss_func(outputs, labels)

    # Zero the parameter gradients, backward pass, optimize
    optimizer.zero_grad()
    train_loss.backward()
    optimizer.step()

    # Print progress
    if epoch % (NOF_EPOCHS / 10) == 0:

        model.eval()

        with torch.no_grad():

            # Forward pass on test data
            inputs, labels = X_test_norm.to(DEVICE), Y_test_norm.to(DEVICE)
```

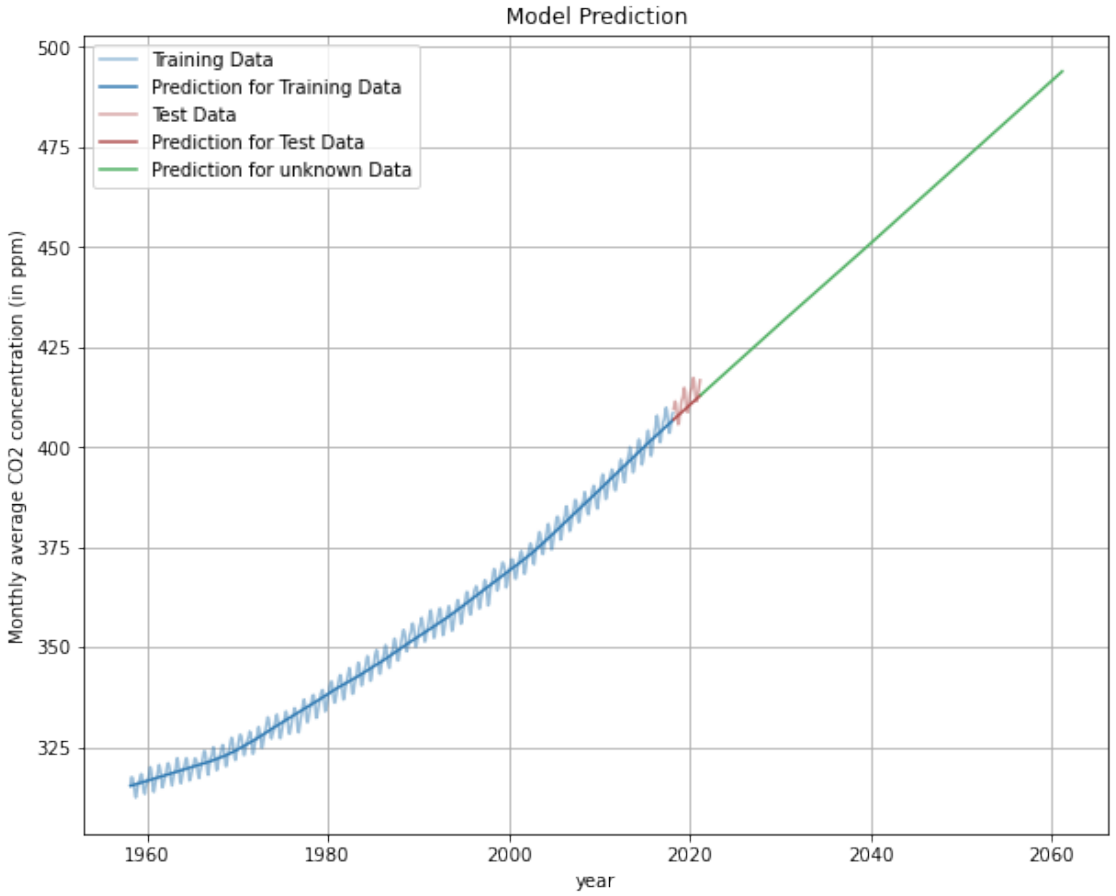
```
            outputs   = model(inputs)
            test_loss = loss_func(outputs, labels)

            print(  f'Epoch {epoch:4};    '
                  + f'Training Loss: {train_loss.item():.3f};    '
                  + f'Test Loss: {test_loss.item():.3f}')
```

```
Network nof_parameters = 301

Training Progress...
Epoch    0;    Training Loss: 1.344;    Test Loss: 6.862
Epoch   50;    Training Loss: 0.049;    Test Loss: 0.231
Epoch  100;    Training Loss: 0.019;    Test Loss: 0.063
Epoch  150;    Training Loss: 0.009;    Test Loss: 0.023
Epoch  200;    Training Loss: 0.007;    Test Loss: 0.013
Epoch  250;    Training Loss: 0.007;    Test Loss: 0.011
Epoch  300;    Training Loss: 0.007;    Test Loss: 0.011
Epoch  350;    Training Loss: 0.007;    Test Loss: 0.012
Epoch  400;    Training Loss: 0.007;    Test Loss: 0.012
Epoch  450;    Training Loss: 0.006;    Test Loss: 0.012
```

[8]:
```
# Evaluate trained model
model.eval()
model.to('cpu')
with torch.no_grad():
    Y_train_predict = model(X_train_norm) * Y_scale + Y_shift
    Y_test_predict  = model(X_test_norm)  * Y_scale + Y_shift
    Y_pred_predict  = model(X_pred_norm)  * Y_scale + Y_shift
```

[9]:
```
# Visualization
fig, ax = plt.subplots(1, 1, figsize=(10, 8))

# Training data
ax.plot(X_train, Y_train,
        color=col_train, label='Training Data', alpha=0.5)
ax.plot(X_train, Y_train_predict,
        color=col_train, label='Prediction for Training Data')

# Test data
ax.plot(X_test,  Y_test,
        color=col_test, label='Test Data', alpha=0.5)
ax.plot(X_test, Y_test_predict,
        color=col_test, label='Prediction for Test Data')

# Pred data
ax.plot(X_pred, Y_pred_predict,
```

```
       color=col_pred, label='Prediction for unknown Data')

# General plot settings
ax.set_xlabel('year')
ax.set_ylabel('Monthly average CO2 concentration (in ppm)')
ax.set_title('Model Prediction')
ax.grid()
ax.legend()
plt.show()
```



[ ]: