Solution

May 3, 2022

Julian Zimmerlin 6009977
Leander Zimmermann 4165446

Probabilistic Machine Learning

Machine Learning in Science, University of Tübingen, Summer Semester 2022

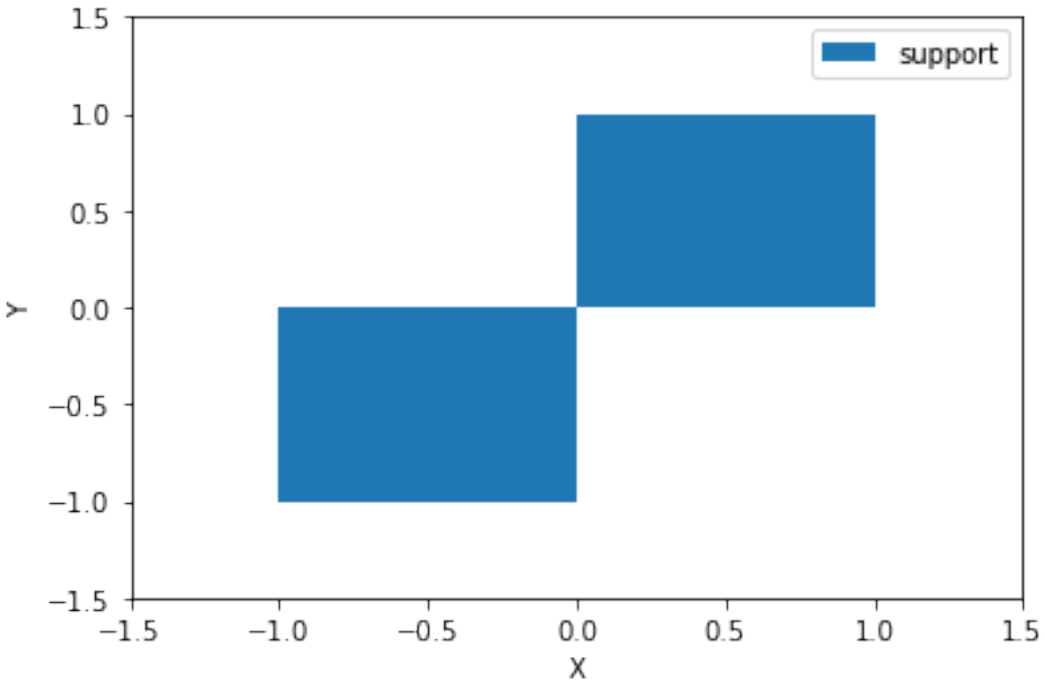
1 Exercise 02

hand in before 06.05.2022, 12:00 p.m. (noon)

1.1 EXAMple

1.1.1 a)

[17]:
```
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
fig, ax = plt.subplots()
ax.add_patch(Rectangle((0,0), 1, 1))
ax.add_patch(Rectangle((-1,-1), 1,1))
ax.set_ylim(-1.5,1.5)
ax.set_xlim(-1.5,1.5)
ax.set_ylabel("Y")
ax.set_xlabel("X")
ax.legend(["support"])
plt.show()
```



1.1.2 b)

They are not independent, as $P(Y \mid X = -0.5) \neq P(Y \mid X = 0.5)$, as can be seen from the plot above

1.1.3 c)

$$\begin{aligned}
\iint p(x,y)dxdy &= \int_{-1}^{1}\left(\int_{-1}^{0} p(x,y)dx + \int_{0}^{1} p(x,y)dx\right)dy \\
&= \int_{-1}^{0}\int_{-1}^{0} p(x,y)dxdy + \underbrace{\int_{0}^{1}\int_{-1}^{0} p(x,y)dxdy}_{=0} + \int_{0}^{1}\int_{0}^{1} p(x,y)dxdy + \underbrace{\int_{-1}^{0}\int_{0}^{1} p(x,y)dxdy}_{=0} \\
&= \int_{-1}^{0}\int_{-1}^{0} \frac{1}{2}dxdy + \int_{0}^{1}\int_{0}^{1} \frac{1}{2}dxdy \\
&= \frac{1}{2} + \frac{1}{2} = 1
\end{aligned}$$

1.1.4 d)

$$\int_{-1}^{0} p(x,y)dy = \begin{cases} \frac{1}{2} & \text{if } -1 < x < 0 \\ 0 & \text{else} \end{cases}$$

$$\int_{0}^{1} p(x,y)dy = \begin{cases} \frac{1}{2} & \text{if } 0 < x < \\ 0 & \text{else} \end{cases}$$

$$\begin{aligned} p(x) &= \int p(x,y)dy \\ &= \int_{-1}^{0} p(x,y)dy + \int_{0}^{1} p(x,y)dy \\ &= \begin{cases} \frac{1}{2} & \text{if } -1 < x < 0 \\ \frac{1}{2} & \text{if } 0 < x < -1 \\ 0 & \text{else} \end{cases} \end{aligned}$$

By symmetry: $p(x) = p(y)$

1.1.5 e)

$$\begin{aligned} p(y \mid x) &= \frac{p(x,y)}{p(x)} \\ &= \begin{cases} \frac{0.5}{0.5} & \text{if } -1 < x < 0, -1 < y < 0 \\ \frac{0.5}{0.5} & \text{if } 0 < x < -1, 0 < y < 1 \\ 0 & \text{else} \end{cases} \\ &= \begin{cases} 1 & \text{if } -1 < x < 0, -1 < y < 0 \\ 1 & \text{if } 0 < x < -1, 0 < y < 1 \\ 0 & \text{else} \end{cases} \\ &= 2p(x,y) \end{aligned}$$

By symmetry: $p(x \mid y) = p(y \mid x)$

1.1.6 f)

$$E(X) = \int_{-1}^{0} \frac{1}{2} x dx + \int_{0}^{1} \frac{1}{2} x dx = 0$$

$$\begin{aligned} E(X \mid Y) &= \int p(x \mid y) x dx \\ &= \begin{cases} \int_{0}^{1} x dx & \text{if } 0 < y < 1 \\ \int_{-1}^{0} x dx & \text{if } -1 < y < 0 \end{cases} \\ &= \begin{cases} \frac{1}{2} & \text{if } 0 < y < 1 \\ -\frac{1}{2} & \text{if } -1 < y < 0 \end{cases} \end{aligned}$$

1.1.7 g)

$$E(X^2) = \int_{-1}^{0} \frac{1}{2}x^2 dx + \int_{0}^{1} \frac{1}{2}x^2 dx = \frac{1}{3}$$
$$Var(X) = E(X^2) - E(X)^2 = \frac{2}{6}$$

1.1.8 h)

$$\begin{aligned}
Cov(X,Y) &= E(XY) - E(X)E(Y) = E(XY) \\
&= \iint p(x,y)xy \; dxdy \\
&= \int_{-1}^{1} \left(\int_{-1}^{0} p(x,y)xy \; dx + \int_{0}^{1} p(x,y)xy \; dx \right) dy \\
&= \int_{-1}^{0}\int_{-1}^{0} p(x,y)xy \; dxdy + \underbrace{\int_{0}^{1}\int_{-1}^{0} p(x,y)xy \; dxdy}_{=0} + \int_{0}^{1}\int_{0}^{1} p(x,y)xy \; dxdy + \underbrace{\int_{-1}^{0}\int_{0}^{1} p(x,y)xy \; dxdy}_{=0} \\
&= \int_{-1}^{0}\int_{-1}^{0} \frac{1}{2}xy \; dxdy + \int_{0}^{1}\int_{0}^{1} \frac{1}{2}xy \; dxdy \\
&= \int_{-1}^{0} \frac{1}{2}x^2 \; dx + \int_{0}^{1} \frac{1}{2}x^2 \; dx \\
&= \frac{1}{3}
\end{aligned}$$

1.1.9 i)

$$Z = e^{-X}$$
$$p(Z) = \begin{cases} \frac{1}{2} & \text{if } 0 < x < 1 \\ -\frac{1}{2} & \text{if } -1 < x < 0 \\ 0 & \text{else} \end{cases}$$
$$= \begin{cases} \frac{1}{2} & \text{if } 1 > z > \frac{1}{e} \\ -\frac{1}{2} & \text{if } e > z > 1 \\ 0 & \text{else} \end{cases}$$

1.2 Theory

$$
\begin{aligned}
p(t) &= \mathcal{N}(t \mid 10, 3^2) \\
p(t_A \mid t) &= \mathcal{N}(t_A \mid t, 1) \\
p(t_B \mid t) &= \mathcal{N}(t_B \mid 2t + 3, 2^2) \\
\text{or centered to t:} & \\
p(t_B \mid t) &= \mathcal{N}(\frac{t_B - 3}{2} \mid t, 1) \\
\text{with these we know from the lecture notes} & \\
\text{because } t_A \perp\!\!\!\perp t_B \mid t & \\
p(t \mid t_A, t_B) &= \mathcal{N}(t \mid m, s^2) \\
\text{with} & \\
\frac{1}{s^2} &= \frac{1}{3^2} + \frac{1}{1} + \frac{1}{1} \\
&= \frac{19}{9} \\
s^2 &= \frac{9}{19} \\
m &= \left(\frac{1}{3^2} \cdot 10 + \frac{1}{1} \cdot 10 + \frac{1}{1} \cdot \frac{18-3}{2}\right) \cdot \frac{9}{19} \\
&= \left(\frac{10}{9} + \frac{10}{1} + \frac{15}{2}\right) \cdot \frac{9}{19} \\
&= \left(\frac{20}{18} + \frac{180}{18} + \frac{135}{18}\right) \cdot \frac{9}{19} \\
&= \frac{335}{18} \cdot \frac{9}{19} \\
&= \frac{335}{38} \approx 8.816 \\
\implies & \\
p(t \mid t_A, t_B) &= \mathcal{N}\left(t \,\middle|\, \frac{335}{38}, \frac{9}{19}\right)
\end{aligned}
$$

1.3 Deep Learning on the *Keeling curve*

Adapted from Prof. Dr. Philipp Hennig, Emilia Magnani & Lukas Tatzel.

In previous lecture courses you have become accustomed with various models of *Deep Learning*. In popular texts it can sometimes sound as if deep learning has made all other concepts of machine learning obsolete. The point of this exercise is to reflect on this sentiment.

You will work with the famous Keeling curve, a time series of atmospheric CO2 concentrations collected at the Mauna Loa observatory in Hawaii. Your task is to produce an extrapolation of this dataset from today 40 years into the future, using a deep learning framework. This notebook will download the data and train a basic neural network using pytorch.

[29]:
```
from matplotlib import pyplot as plt
%matplotlib inline

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

1.3.1 1) Load Data

As a first step, let's load the CO2 data into a pandas dataframe.

[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
..    …    …          …     …
764  2021     11    2021.8750  415.01
765  2021     12    2021.9583  416.71
766  2022      1    2022.0417  418.19
767  2022      2    2022.1250  419.28
768  2022      3    2022.2083  418.81

[769 rows x 4 columns]
```

1.3.2 2) Split Data

Now, we split the data into training and test data.

[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)
```

1.3.3 3) Preprocess Data

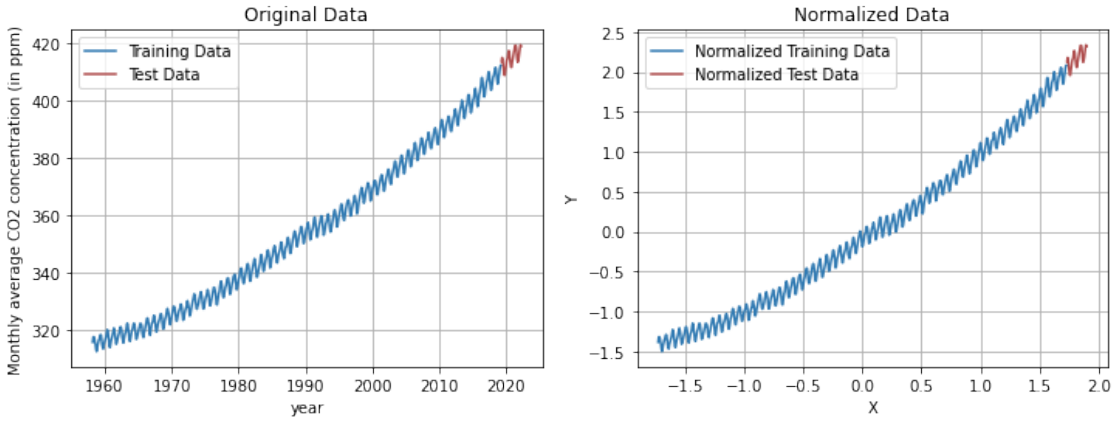
Let's normalize our data.

[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()
```



1.3.4 4) Train the Neural Network

Define the model architecture, the loss function and the optimizer. Use the normalized data to train the neural network for a given number of epochs.

[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")
print(f'Using device: {DEVICE}')

# 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}')
```

```
Using device: cpu
Network nof_parameters = 301

Training Progress…
Epoch    0;    Training Loss: 1.344;    Test Loss: 6.872
Epoch   50;    Training Loss: 0.050;    Test Loss: 0.234
Epoch  100;    Training Loss: 0.018;    Test Loss: 0.064
Epoch  150;    Training Loss: 0.009;    Test Loss: 0.023
Epoch  200;    Training Loss: 0.007;    Test Loss: 0.013
Epoch  250;    Training Loss: 0.006;    Test Loss: 0.011
Epoch  300;    Training Loss: 0.006;    Test Loss: 0.011
Epoch  350;    Training Loss: 0.006;    Test Loss: 0.011
Epoch  400;    Training Loss: 0.006;    Test Loss: 0.012
Epoch  450;    Training Loss: 0.006;    Test Loss: 0.012
```

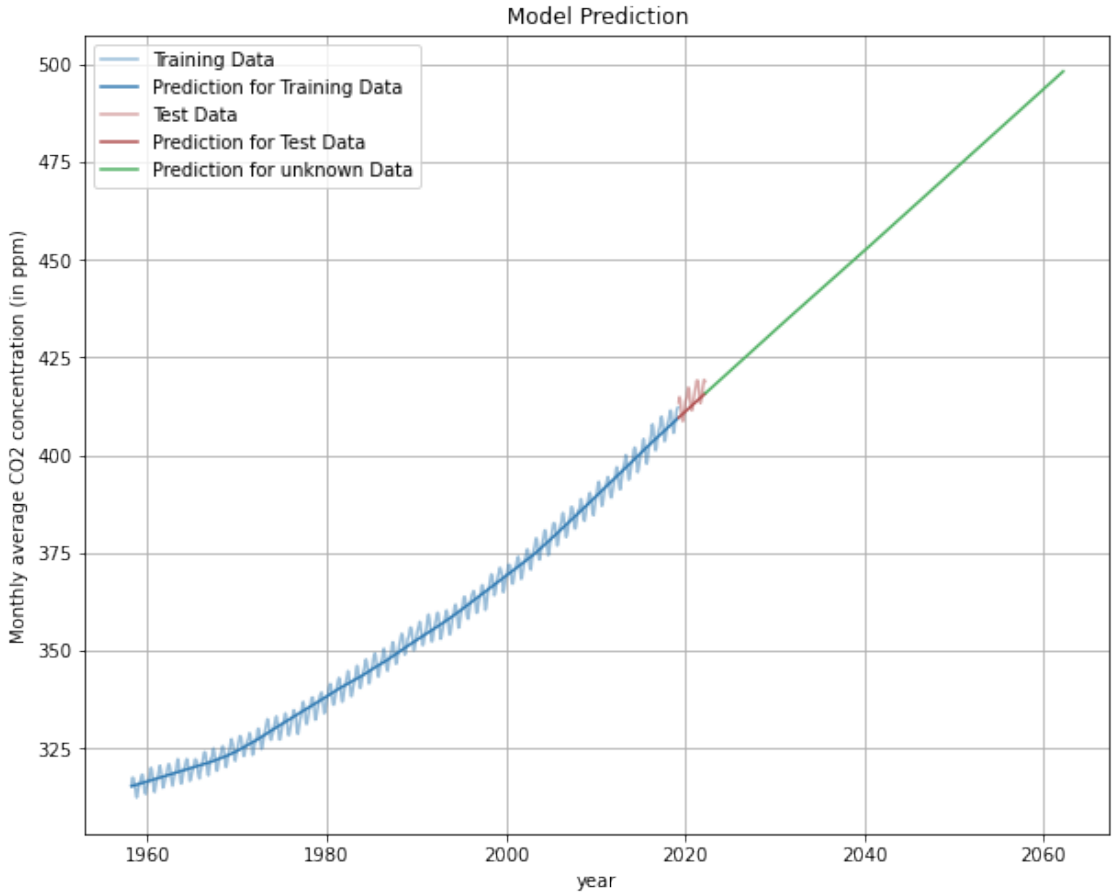
1.3.5 5) Evaluate Trained Model

Let's explore the model extraplolation.

[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()
```



1.3.6 Your Tasks

First, provide answers to the following two questions.

(1) Are you happy with the extrapolation? Why/why not?

Answer: Not really. While the model picks up the general upward trend, it fails to capture the yearly cycle of CO2 concentration (i.e., the oscillatory component in the data).

---

(2) Do you trust the extrapolation? Why/why not?

Answer: Not really. 40 years is a long time. I think that current climate policy decisions will heavily impact the CO2 levels in 40 years. Perhaps the extrapolation can give us an idea what would happen if contries stay with their current climate policies (i.e. CO2 levels will continue to increase).

Task: Your task is to come up with your own solution using a deep learning framework of your choice. First, develop an alternative concept to compute the extrapolation. Write down your idea and briefly describe why and in what way your approach is expected to be an improvement compared to the extrapolation above. A few lines of text should be enough. Note however, that e.g. just adding layers and changing the optimizer won't be regarded as "sufficient". We want to see that you really put some thought into your concept.

Your concept: To model the oscillatory component, we will add a feature to the input data that represents the month. We can do this easily because we know that the oscillatory component has phase one in the unnormalized data (because it is caused by the seasons, presumably because of ice melting in the summer, releasing CO2). This should enable the model to learn the oscillatory component better and thus improve train error, test error and generalization performance.

Task: Actually implement your idea. You can use the above code as a template. Of course, you are free to re-organize the data, modify the network architecture, the optimizer, loss function etc. You can also use a different framework than pytorch, if you want.

[108]:
```
def add_month_info(dataset_list):
    for dataset in dataset_list:
        phase = torch.arange(0,12).repeat(dataset.size()[0]//12+1)  # on the␣
↪unnormalized data the period length would be 1 (= 1 year)
        phase = phase[:dataset.size()[0], np.newaxis] / 11
        yield torch.cat((dataset, phase), axis=1)
```

[109]:
```
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

# Add phase info
```

```
[X_train_norm_phase, X_test_norm_phase, X_pred_norm_phase] =␣
 ↪add_month_info([X_train_norm, X_test_norm, X_pred_norm])
print(X_train_norm_phase.size())

# 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
```

```
torch.Size([733, 2])
```

[120]:
```
# 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")
print(f'Using device: {DEVICE}')

# Choose neural network architecture
model = nn.Sequential(
    nn.Linear(2, 100),
    nn.ReLU(),
    nn.Linear(100, 50),
    nn.ReLU(),
    nn.Linear(50, 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(), weight_decay=0.0001)

# Train for NOF_EPOCHS epochs
NOF_EPOCHS = 1000

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

    model.train()

    # Forward pass
    inputs, labels = X_train_norm_phase.to(DEVICE), Y_train_norm.to(DEVICE)
```

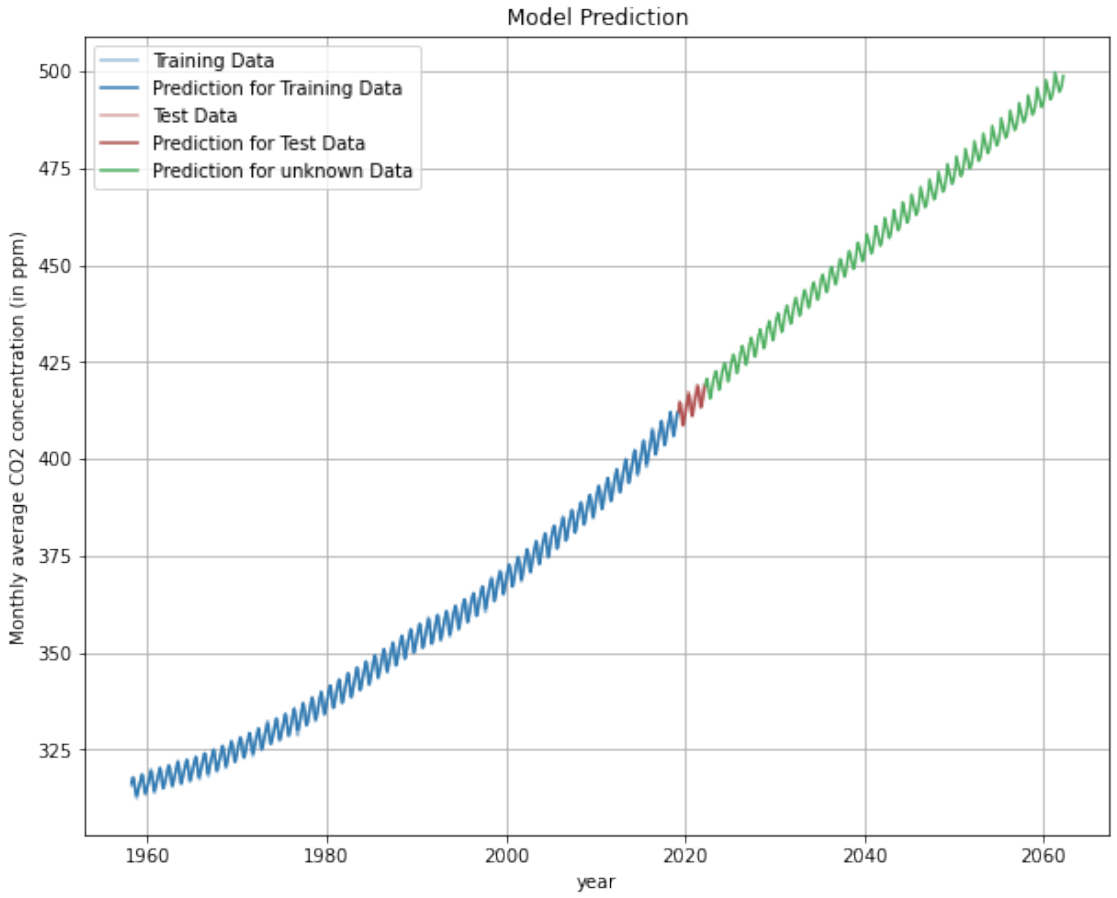
```
    outputs = model(inputs)
    train_loss = loss_func(outputs, labels)

    # 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_phase.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}')
```

```
Using device: cpu
Network nof_parameters = 5401

Training Progress…
Epoch    0;    Training Loss: 0.975;    Test Loss: 4.112
Epoch  100;    Training Loss: 0.004;    Test Loss: 0.018
Epoch  200;    Training Loss: 0.001;    Test Loss: 0.005
Epoch  300;    Training Loss: 0.000;    Test Loss: 0.003
Epoch  400;    Training Loss: 0.000;    Test Loss: 0.003
Epoch  500;    Training Loss: 0.000;    Test Loss: 0.003
Epoch  600;    Training Loss: 0.000;    Test Loss: 0.003
Epoch  700;    Training Loss: 0.000;    Test Loss: 0.002
Epoch  800;    Training Loss: 0.000;    Test Loss: 0.002
Epoch  900;    Training Loss: 0.000;    Test Loss: 0.002
```

[121]:
```
# Evaluate trained model
model.eval()
model.to('cpu')
with torch.no_grad():
    Y_train_predict = model(X_train_norm_phase) * Y_scale + Y_shift
    Y_test_predict  = model(X_test_norm_phase)  * Y_scale + Y_shift
```

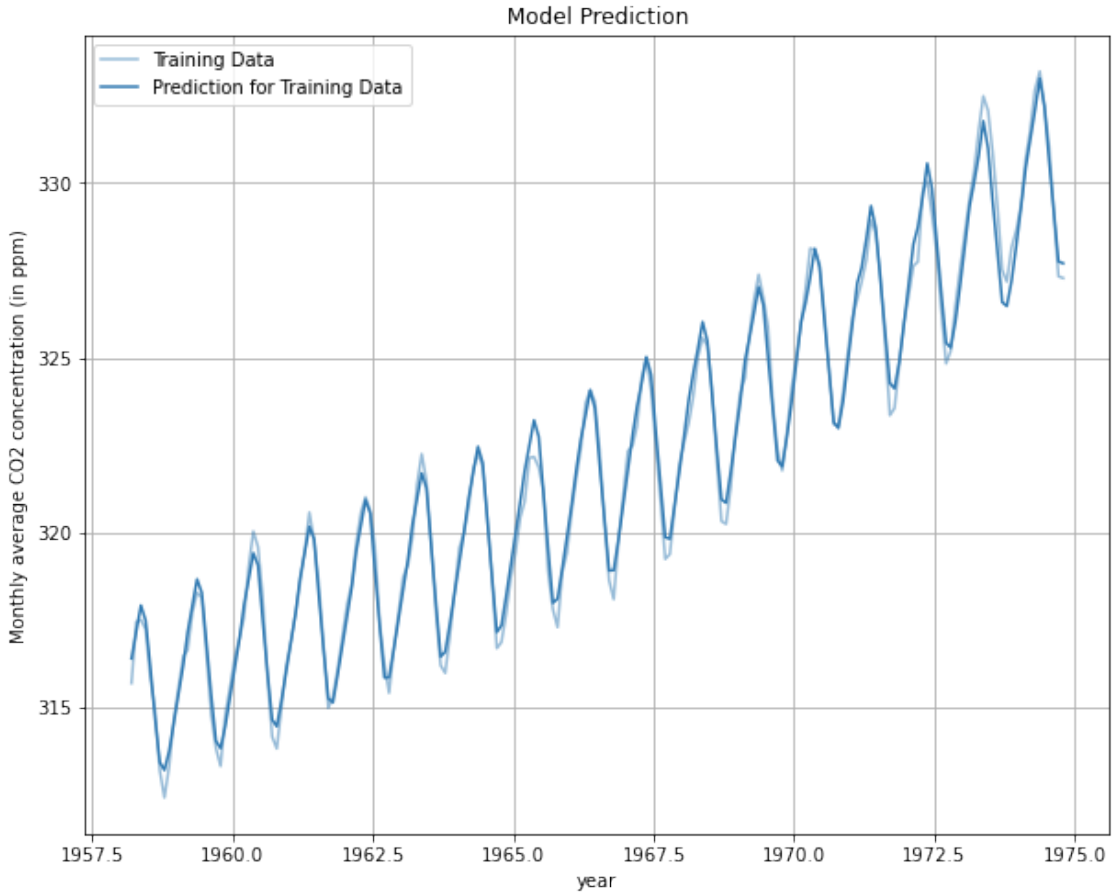
```
    Y_pred_predict  = model(X_pred_norm_phase)  * Y_scale + Y_shift
```

[122]:
```
# 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()
```



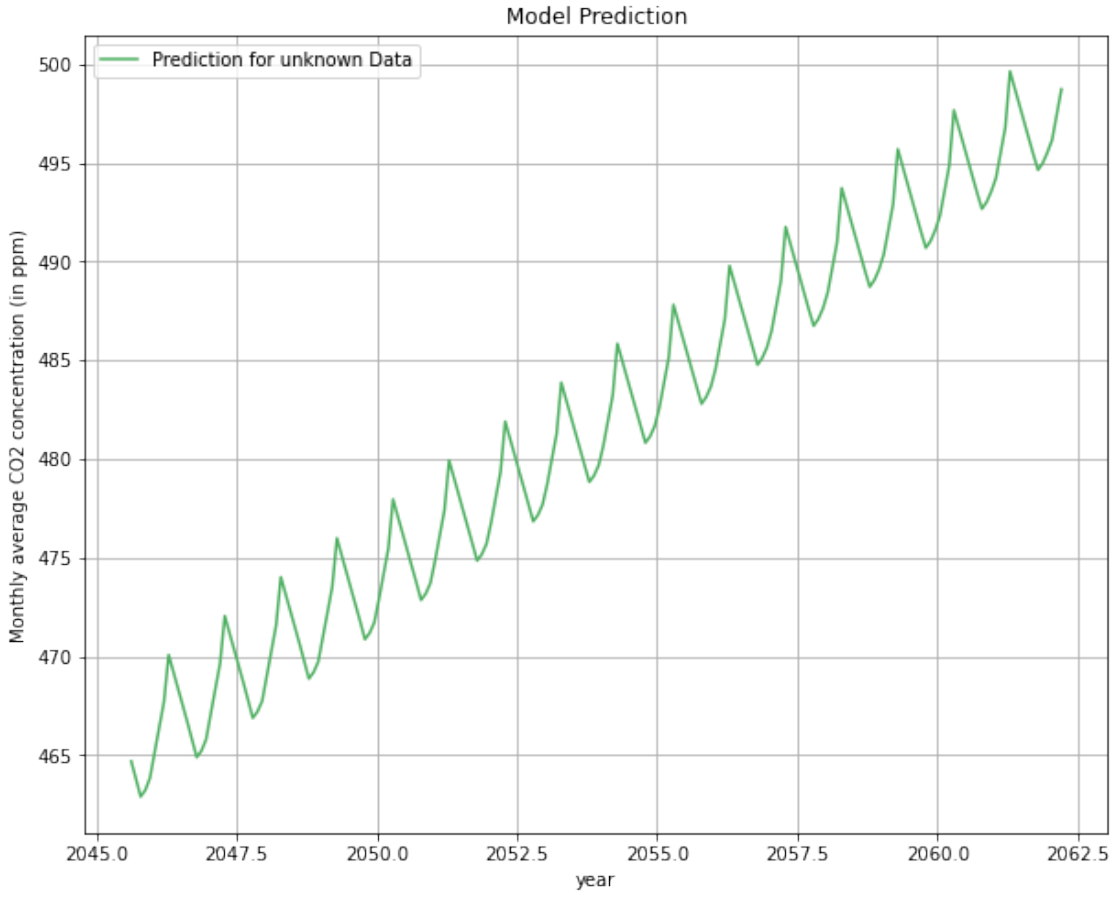
[123]:
```
# Plot just train data

num_pts = 200

# Visualization
fig, ax = plt.subplots(1, 1, figsize=(10, 8))

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



[124]:
```
# Plot just pred data

num_pts = 200

# Visualization
fig, ax = plt.subplots(1, 1, figsize=(10, 8))

# Pred data
ax.plot(X_pred[-num_pts:], Y_pred_predict[-num_pts:],
       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()
```



1.4 Comment

As expected, adding phase info (= the month) enabled the network to capture the oscillatory nature of the data better. Note that we also added another layer to the network. While it also sort of worked without the extra layer, we found that the network still had some problems approximating the sinusoidal shape (presumably it is simply hard to model a sine wave with a single ReLU layer - a piecewise linear function). In addition, we added weight decay ($\lambda = 0.0001$) because otherwise we experienced a form of overfitting where the sinusoidal shape would get distorted when extrapolating into the future.