Exercise 01

April 24, 2023

1 Probabilistic Machine Learning

University of Tübingen, Summer Term 2023 © 2023 P. Hennig

Exercise Sheet No. 1 — Setting Up

Submission by:

- Batuhan, Oezcoemlekci, Matrikelnummer: 6300476
- Aakarsh, Nair, Matrikelnummer: 6546577

1.1 Exercise 1.1

Assume that, if A is true, B becomes more plausible. That is, $P(B|A) \geq P(B)$.

Using the rules of probability (sum rule and product rule) as stated in the lecture, show the following relationships (stated in the lecture without proof)

NOTE that sum and product rule implies the Bayes' Rule. Also note that any probability is between [0,1].

(a) $P(B|\neg A) \leq P(B)$ **("If A is false, B becomes less plausible")**

$$P(B|A) \geq P(B)$$

$$P(B,A)/P(A) \geq P(B)$$

$$P(A|B)P(B)/P(A) \geq P(B)$$

$$1 - P(A|B) \leq 1 - P(A)$$

$$P(\neg A|B) \leq P(\neg A)$$

$$P(B|\neg A)P(\neg A)/P(B) \leq P(\neg A)$$

$$P(B|\neg A)/P(B) \leq 1$$

$$P(B|\neg A) \leq P(B)$$

$$QED$$

Longer Variation

We have been given that $P(B|A) \geq P(B)$, we start by exploring the consequences of this.

Using the *law of total probability* we know that:

$$P(B) = P(A)P(B|A) + (1 - P(A))P(B|\neg A)$$

Where we additionally used that $P(\neg A) = (1 - P(A))$.

Substituting in our given inequality we see that:

$$P(B|A) \geq P(A)P(B|A) + (1 - P(A))P(B|\neg A)$$

Simplifying we get the inequality:

$$(1 - P(A))P(B|A) \geq (1 - P(A))P(B|\neg A)$$

We assume $P(A) \neq 1$, and we know that $P(A) \leq 1$, by virtue of being a probability distribution.

Thus dividing by $(1 - P(A))$ on both sides leaves the inequality unaltered, we get:

$$P(B|A) \geq P(B|\neg A)$$

or $P(B|A) - P(B|\neg A) \geq 0$

Now we revisit the law of total probability:

$$P(B) = P(B|A)P(A) + (1 - P(A))P(B|\neg A)$$

simplifying we get

$$P(B) = P(B|A)P(A) - P(B|\neg A)P(A) + P(B|\neg A)$$

Consider the first two terms combined as $P(A)(P(B|A) - P(B|\neg A))$. From before we have \$P(A) 0 \$ and $P(B|A) - P(B|\neg A) \geq 0$. Thus the combined term $P(A)(P(B|A) - P(B|\neg A)) \geq 0$. Let \$ = P(A) (P(B|A) - P(B| ¬A)) \$ then we have $\delta \geq 0$.

In our expression we have

$$P(B) = \delta + P(B|\neg A), \delta \geq 0$$

Giving us the inequality we need, $P(B) \geq P(B|\neg A)$

(b) $P(A|B) \geq P(A)$ **("If B is true, A becomes more plausible")**

$$P(B|A) \geq P(B)$$

$$P(B, A)/P(A) \geq P(B)$$

$$P(A|B)P(B)/P(A) \geq P(B)$$

$$P(A|B)P(B) \geq P(B)P(A)$$

$$P(A|B) \geq P(A)$$

$$QED$$

Longer Variation

We have been given that, $P(B|A) \geq P(B)$, that is if A is true then B, becomes more plausible. We can use Bayes' Theorem to rewrite $P(B|A)$ in terms of $P(A|B)$ we note the two are related as

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

We can substitute this expression for $P(B|A)$ in our given inequality $P(B|A) \geq P(B)$ we get:

$$\frac{P(A|B)P(B)}{P(A)} \geq P(B)$$

Multiply both sides of our given inequality by $\frac{P(A)}{P(B)}$, assuming $P(B) \neq 0$, this fraction is positive as both values probabilities.

This gives us:

$$P(A|B) \geq P(A)$$

(c) $P(A|\neg B) \leq P(A)$ **("If B is false, A becomes less plausible")** Starting from the result of (b):

$$P(A|B) \geq P(A)$$
$$P(A, B)/P(B) \geq P(B)$$
$$P(B|A)P(A)/P(B) \geq P(A)$$
$$1 - P(B|A) \leq 1 - P(B)$$
$$P(\neg B|A) \leq P(\neg B)$$
$$P(A|\neg B)P(\neg B)/P(A) \leq P(\neg B)$$
$$P(A|\neg B)/P(A) \leq 1$$
$$P(A|\neg B) \leq P(A)$$
$$QED$$

Longer Version

From the previous questions we have that , given $P(B|A) \geq P(B)$, $P(A|B) \geq P(A)$, thus our argument for $P(A|\neg B) \leq P(B)$ is symmetric to the argument in the first problem but with the variables A and B switched. We only reproduce it here for completeness.

Consider the expression $P(A|B) \geq P(A)$, from the law of total probability for $P(A)$ we have:

$$P(A|B) \geq P(A|B)P(B) + P(A|\neg B)(1 - P(B))$$

We additionally used $P(\neg B) = 1 - P(B)$.

Thus factoring and rearranging the terms we get.

$$P(A|B)(1-P(B)) \geq P(A|\neg B)(1-P(B))$$

Assuming $ P(B) 1 $, we also know that $P(B) \geq 0$, we get.

The previous inequality simplifies to :

$$P(A|B) \geq P(A|\neg B)$$

Thus let $\gamma = P(A|B) - P(A|\neg B)$ then we have $\gamma \geq 0$.

Revisiting the law of total probability for $P(A)$, we have

$$P(A) = P(A|B)P(B) + P(A|\neg B)(1-P(B))$$

Re-arranging the terms we have

$$P(A) = P(B)(P(A|B) - P(A|\neg B)) + P(A|\neg B)$$

As both $P(B) 0 $ and $ = P(A|B) - P(A|\neg B) 0 $ we thus have $P(B) 0 $

$$P(A) = P(B)\gamma + P(A|\neg B)$$

Thus $P(A|\neg B) \leq P(A)$.

Additionally show that probabilistic reasoning includes Boolean logic as a special case, by showing that if A B is interpreted as equivalent to P(B | A) = 1, then the following three statements hold:

(d) $P(\neg A|\neg B) = 1$ **("modus tollens")** We start with the given statement

$$P(B|A) = 1$$

from the law of total probability for binary variables A, B we have:

$$P(B|A) + P(\neg B|A) = 1$$

Thus $P(\neg B|A) = 0$.

However we can write an expression for the term $P(A|\neg B)$ in terms of $P(\neg B|A)$ using bayes theorem as follows:

$$P(A|\neg B) = \frac{P(\neg B|A)P(A)}{P(\neg B)}$$

It follows then that $P(A|\neg B) = 0$.

From the law of total probability we know that:

$$P(\neg A|\neg B) + P(A|\neg B) = 1$$

Thus it follows that as $P(A|\neg B) = 0$ that $P(\neg A|\neg B) = 1$.

(e) $P(B|\neg A) \leq P(B)$ **("If A is false, B becomes less plausible")** We know from the law of probability that

$$P(B) = P(B|A)P(A) + P(B|\neg A)P(\neg A)$$

As shown previously we can rewrite the above expression as

$$P(B) = P(A)(P(B|A) - P(B|\neg A)) + P(B|\neg A)$$

If we let $\delta = P(B|A) - P(B|\neg A)$ we have been given that $P(B|A) = 1$, thus if we can show that $P(B|\neg A) = 0$ then $\delta = 1$.

Substituting the expression $P(\neg A) = 1 - P(A)$ in the above equation and expanding out the equation we see that:

$$P(\neg B|\neg A) + P(B|\neg A) = 1$$

Thus we have $P(B|\neg A) = 0$. Substituting in the equation from total probability of B we get:

$$P(B) = P(A)(1 - 0) + P(B|\neg A)$$

As $P(A) \geq 0$ from the fact that $P(A)$ is a probability distribution,
we have $P(B) \geq P(B|\neg A)$ which can be restated as $P(B|\neg A) \leq P(B)$.

(f) $P(A|B) \geq P(A)$ **("if B is true, A becomes more plausible")** We have been given that $P(B|A) = 1$, thus we can write an expression for $P(A|B)$ as follows:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{P(A)}{P(B)}$$

Assuming that $P(B) \neq 0$ as we know that $P(B) \in [0, 1]$ we know that

$$P(A) \leq \frac{P(A)}{P(B)}$$

in other words we have $P(A) \leq P(A|B)$, that is $P(A|B) \geq P(A)$.

[ ]:
```
from io import StringIO

import pandas as pd
import requests
from jax import numpy as jnp
from matplotlib import pyplot as plt
from matplotlib import ticker

from tueplots import bundles
from tueplots.constants.color import rgb
```

```
plt.rcParams.update(bundles.beamer_moml())
plt.rcParams.update({"figure.dpi": 200})
```

1.2 Exercise 1.2 (Coding Exercise)

This week's exercise serves as an opener for the course. As a first step, make sure the cell above this works for you. This is mostly to ensure you have `jax` and `tueplots` installed in your local environment, which we will use across the course. If you get errors, try a `pip install jax tueplots pandas`. Consider the well-known Mauna Loa CO_2 dataset, which we download directly from NOAA:

[ ]:
```
url = "https://gml.noaa.gov/webdata/ccgg/trends/co2/co2_weekly_mlo.csv"
s = requests.get(url).text

df = pd.read_csv(StringIO(s), sep=",", header=51, na_values="-999.99").dropna()
X = df["decimal"]
Y = df["average"]
```

We do a *very* quick and dirty linear regression on this univariate timeseries.

[ ]:
```
phi = jnp.stack([jnp.ones(len(X)), jnp.asarray(X)])
w = jnp.linalg.solve((phi @ phi.T), phi @ Y)
Ypred_LinReg = w[0] + w[1] * X


def loss(Yhat):
    return jnp.sqrt(((Y - Yhat) ** 2).mean())


print(f"simple linear regression has train RMSE {loss(Ypred_LinReg):0.2f}")
```

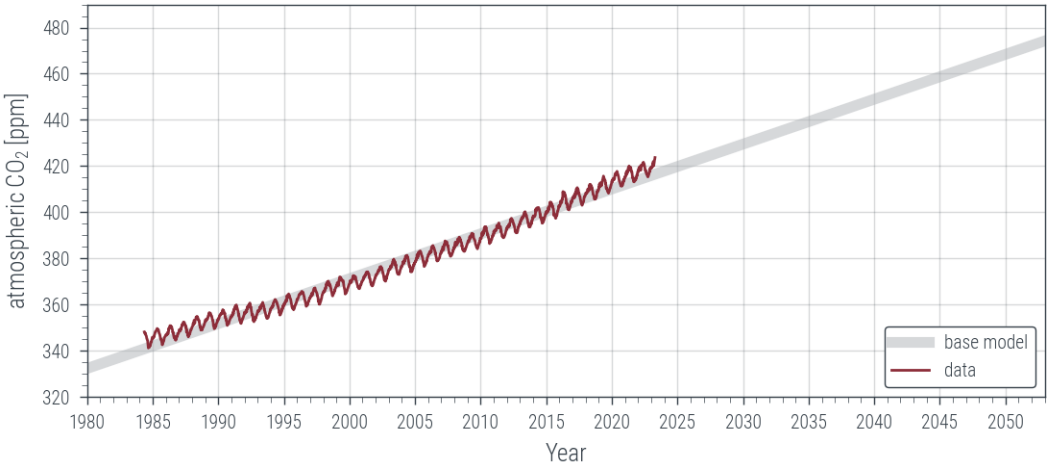
```
WARNING:jax._src.lib.xla_bridge:No GPU/TPU found, falling back to CPU. (Set
TF_CPP_MIN_LOG_LEVEL=0 and rerun for more info.)
```

```
simple linear regression has train RMSE 3.09
```

And make a nice plot:

[ ]:
```
fig, ax = plt.subplots()
x = jnp.linspace(1980, 2053, 200)
ax.plot(x, w[0] + w[1] * x, color=rgb.tue_gray,
        lw=4, alpha=0.5, label="base model")
ax.plot(X, Y, label="data")

ax.set_xlabel("Year")
ax.set_ylabel("atmospheric CO$_2$ [ppm]")
ax.xaxis.set_major_locator(ticker.MultipleLocator(5))
ax.xaxis.set_minor_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(20))
```

```
ax.yaxis.set_minor_locator(ticker.MultipleLocator(5))
ax.set_xlim([1980, 2053])
ax.set_ylim([320, 490])
ax.grid(which="major", axis="both")
ax.legend(loc="lower right", framealpha=1, facecolor="w");
```



1.3 Your Task:

Using your favorite deep learning framework, define and train a supervised (regression) model on this dataset `X,Y`. Use it to predict on the test inputs `x = jnp.linspace(1980, 2053, 200)` from above, i.e. 30 years into the future. Consider the following aspects: * Does your model interpolate between the data (i.e. does it predict the actual pairs (x_i, y_i) correctly, or just approximately)? * Which structural aspects of the data does your model capture (e.g., does its extrapolation also capture the seasonal oscillations present in the data? * Do you know a way to add a form of *confidence* or *uncertainty* to the prediction? Can you make *worst/best-case* and *most likely* predictions for atmospheric CO_2 over the next 30 years?

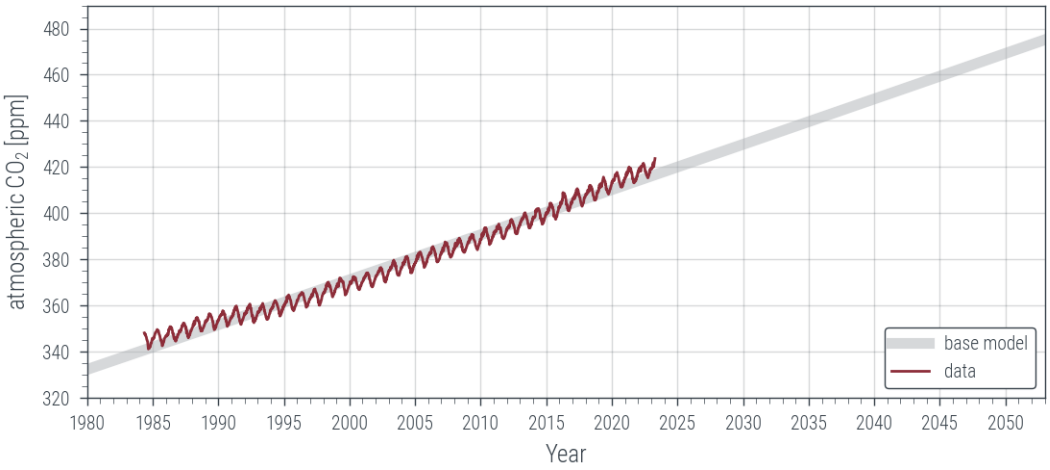
Note: There is obviously no unique "correct" answer to this task. The goal of this exercise is to allow you to reflect on what you may have heard in previous courses about state-of-the-art machine learning models and their training. Deep learning is often perceived as having replaced all of machine learning, and it certainly permeates most of it. However, while deep learning models have led to impressive results on applications like natural language processing and computer vision, they remain nontrivial to train and to interpret; and their extrapolatory behaviour can be underwhelming. Much of the lecture course will be dedicated to understanding models, their training and predictive behaviour better, and to transfer much of what we learn with basic models to deep learning.

Thus, we do not expect you to provide a perfect answer here. Answers will get the "sufficient" mark (i.e. the associated bonus point in the exam) if they * correctly define any model with at least one nonlinearity * train the model such that its RMSE on the train set (as defined above) is *smaller* than that of linear regression. * amend the plot above with the model's prediction on the testset `x` (not the train set!)

1.3.1 Fit a Simple Polynomial of Degree 2

[ ]:
```
# Fit a polynomial of degree 2 to the X data.
phi = jnp.stack([jnp.ones(len(X)), jnp.asarray(X), jnp.asarray(X) ** 2])
w = jnp.linalg.solve((phi @ phi.T), phi @ Y)
Ypred_LinReg = w[0] + w[1] * X + w[2] * X ** 2

# Compute the RMSE.
print(f"simple polynomial of degree 2 regression has train RMSE␣
 ↪{loss(Ypred_LinReg):0.2f}")

# Plot the data and the polynomial.
fig, ax = plt.subplots()
# Change figure size
#fig.set_size_inches(10, 3)
x = jnp.linspace(1980, 2053, 200)
ax.plot(x, w[0] + w[1] * x + w[2] * x ** 2, color=rgb.tue_gray,
        lw=4, alpha=0.5, label="base model")
ax.plot(X, Y, label="data")

ax.set_xlabel("Year")
ax.set_ylabel("atmospheric CO$_2$ [ppm]")
ax.xaxis.set_major_locator(ticker.MultipleLocator(5))
ax.xaxis.set_minor_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(20))
ax.yaxis.set_minor_locator(ticker.MultipleLocator(5))
ax.set_xlim([1980, 2053])
ax.set_ylim([320, 490])
ax.grid(which="major", axis="both")
ax.legend(loc="lower right", framealpha=1, facecolor="w");
```

```
simple polynomial of degree 2 regression has train RMSE 3.06
```



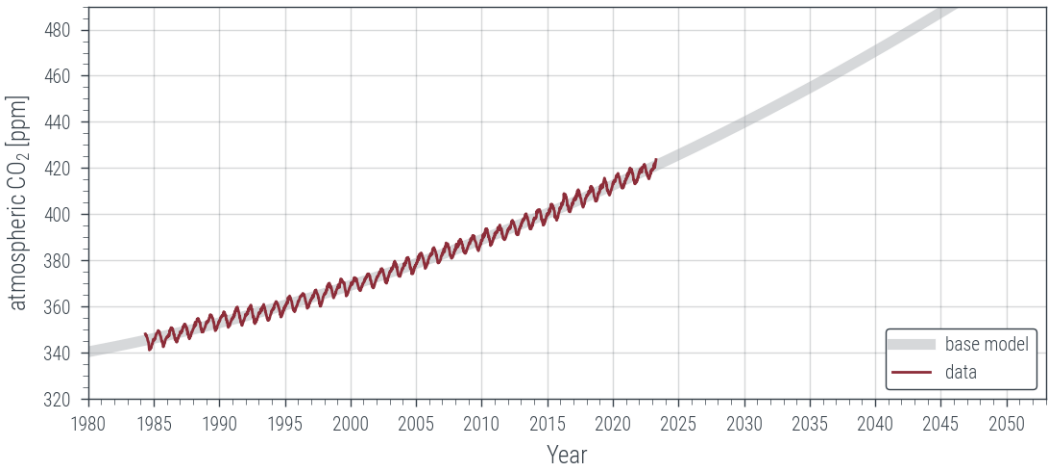
Here nonlinearity is small but present since the training RMSE (3.06) is smaller than that of linear regression (3.09)

1.3.2 Least Square Fit of a Polynomial of Degree 2 (Using a different library)

We obtain better RMSE once we use numpy polyfit to fit second degree polynomial into our data. **(RMSE 2.29<3.09)**

[ ]:
```
import numpy as np

X_np = np.asarray(X)
Y_np = np.asarray(Y)
W = np.polyfit(X_np, Y_np, 2)
p = np.poly1d(W)

Ypred_LinReg = p(X_np)

# Compute the RMSE.
print(f"simple polynomial of degree 2 regression has train RMSE␣
 ↪{loss(Ypred_LinReg):0.2f}")

# Plot the data and the polynomial.
fig, ax = plt.subplots()
# Change figure size
#fig.set_size_inches(10, 3)
x = np.linspace(1980, 2053, 200)
ax.plot(x, p(x), color=rgb.tue_gray,
        lw=4, alpha=0.5, label="base model")
ax.plot(X, Y, label="data")

ax.set_xlabel("Year")
ax.set_ylabel("atmospheric CO$_2$ [ppm]")
ax.xaxis.set_major_locator(ticker.MultipleLocator(5))
ax.xaxis.set_minor_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(20))
ax.yaxis.set_minor_locator(ticker.MultipleLocator(5))
ax.set_xlim([1980, 2053])
ax.set_ylim([320, 490])
ax.grid(which="major", axis="both")
ax.legend(loc="lower right", framealpha=1, facecolor="w");
```

```
simple polynomial of degree 2 regression has train RMSE 2.29
```



1.3.3 Fit a Multi Layer Perceptron

[ ]:
```
# Fit a multilayer perceptron to the X data using PyTorch.
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.optim.lr_scheduler import ExponentialLR
import numpy as np
from IPython.display import clear_output
%matplotlib inline

def visualize(net, X, Y): # PHI_tensor, Y_tensor
    clear_output(wait=True)
    # Plot on the test data
    fig, ax = plt.subplots()
    # Change figure size
    fig.set_size_inches(12, 8)
    x = np.linspace(1980, 2053, 200)
    x_tensor = torch.tensor(x, dtype=torch.float).reshape(-1, 1)
    #phi_tensor = torch.stack([torch.ones(len(x)), x_tensor.squeeze()], dim=1)
    y = net(x_tensor).detach().squeeze().numpy()
    ax.plot(x, y, color=rgb.tue_gray,
            lw=4, alpha=0.5, label="base model")
    ax.plot(X, Y, label="data")

    ax.set_xlabel("Year")
    ax.set_ylabel("atmospheric CO$_2$ [ppm]")
    ax.xaxis.set_major_locator(ticker.MultipleLocator(5))
    ax.xaxis.set_minor_locator(ticker.MultipleLocator(1))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(20))
```

```
    ax.yaxis.set_minor_locator(ticker.MultipleLocator(5))
    ax.set_xlim([1980, 2053])
    ax.set_ylim([320, 490])
    ax.grid(which="major", axis="both")
    ax.legend(loc="lower right", framealpha=1, facecolor="w")

    plt.show();


# Define the model.
class SimpleMLP(nn.Module):
    def __init__(self, input_size=1, hidden_size=128, output_size=1):
        super(SimpleMLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size, dtype=torch.float,␣
 ↪bias=True)
        #self.fc2 = nn.Linear(hidden_size, hidden_size, dtype=torch.float,␣
 ↪bias=True)
        self.fc3 = nn.Linear(hidden_size, output_size, dtype=torch.float,␣
 ↪bias=True)

        #self.fc1.weight.data = torch.tensor(np.random.randn(hidden_size,␣
 ↪input_size), dtype=torch.float)
        #self.fc2.weight.data = torch.tensor(np.random.randn(hidden_size,␣
 ↪hidden_size), dtype=torch.float)
        #self.fc3.weight.data = torch.tensor(np.random.randn(output_size,␣
 ↪hidden_size), dtype=torch.float)



    def forward(self, x):
        x = F.leaky_relu(self.fc1(x))
        #x = F.leaky_relu(self.fc2(x))
        x = self.fc3(x)
        return x

def mse_loss(Y, Yhat):
    if Y.ndim == 2:
        Y_1 = Y.squeeze(1)
    else:
        Y_1 = Y
    if Yhat.ndim == 2:
        Yhat_1 = Yhat.squeeze(1)
    else:
        Yhat_1 = Yhat
    return (((Y_1 - Yhat_1) ** 2).mean())**0.5
```

```
net = SimpleMLP(hidden_size=1, input_size=1)

# Define the optimizer.
optimizer = optim.Adam(net.parameters(), lr=1)
scheduler = ExponentialLR(optimizer, gamma=0.99)

criterion = nn.MSELoss()

# Convert the data to tensors.
X_tensor = torch.tensor(X.values, dtype=torch.float).reshape(1,-1)
#PHI_tensor = torch.stack([torch.ones(len(X)), X_tensor], dim=1)
Y_tensor= torch.tensor(Y.values, dtype=torch.float).reshape(1, -1)

# Train the model.


for epoch in range(500):
    optimizer.zero_grad()
    output = net(X_tensor)
    loss_optim = mse_loss(Y_tensor, output)
    print(f"Epoch {epoch} has loss {loss_optim.item():0.2f}")
    loss_optim.backward()
    optimizer.step()
    if epoch % 5 == 0:
        visualize(net, X, Y)
    if epoch % 50 == 0:
        scheduler.step()

# Compute the RMSE.
Ypred = net(X_tensor).detach().squeeze().numpy()

print(f"simple MLP has train RMS {mse_loss(Y_tensor, Ypred):0.2f}")
```

Cannot regress perfectly, the model is too complex for the given data. It's hard for the MLP weights to converge.

1.3.4 Fit an LSTM Based Prediction

While training the neural network we ran into a problem with training them thus we reconstituted it to try use the LSTM architecture for training problem. We chose a look back duration of 250 weeks and modified the dataset to feed this input sequentially to the LTSM. We find that our model is able over-fit the training data. However extrapolation for regions the training set we do not see we see a market decay in LSTM predictions as the recurrence signal is weak several timesteps away from initial prediction.

[ ]:
```
import torch
import numpy as np
import torch.optim as optim
```

```
import torch.utils.data as data
import torch.nn as nn


def create_dataset(dataset, lookback):
    """Transform a time series into a prediction dataset

    Args:
        dataset: A numpy array of time series, first dimension is the time steps
        lookback: Size of window for prediction
    """
    X, y = [], []
    for i in range(len(dataset)-lookback):
        feature = dataset[i:i+lookback]
        target = dataset[i+1:i+lookback+1]
        X.append(feature)
        y.append(target)
    return torch.tensor(X, dtype=torch.float32).reshape(len(X), lookback, 1),␣
 ↪torch.tensor(y, dtype=torch.float32).reshape(len(y), lookback, 1)


lookback = 250
timeseries = Y.to_numpy()
train_size = int(len(timeseries) * 0.67)
test_size = len(timeseries) - train_size
train, test = timeseries[:train_size], timeseries[train_size:]

X_train, y_train = create_dataset(train, lookback=lookback)
X_test, y_test = create_dataset(test, lookback=lookback)

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)



class LSTM_Model(nn.Module):
    """
    A two layer LTSM model with 52 hidden units is used to predict the next␣
 ↪value in the time series.
    """
    def __init__(self):
        super().__init__()
        self.lstm = nn.LSTM(input_size=1, hidden_size=52, num_layers=2,␣
 ↪batch_first=True, dtype=torch.float32)
        self.linear = nn.Linear(52, 1, dtype=torch.float32)

    def forward(self, x):
```

```
        x, _ = self.lstm(x)
        x = self.linear(x)
        return x


model = LSTM_Model()
model.to('cuda')

optimizer = optim.Adam(model.parameters())
loss_fn = nn.MSELoss()
gpu_loss_fn = nn.MSELoss().to('cuda')
# Move inputs and targets to GPU
X_train = X_train.to('cuda')
y_train = y_train.to('cuda')
X_test = X_test.to('cuda')
y_test = y_test.to('cuda')
# Move loss function to GPU
loader = data.DataLoader(data.TensorDataset(X_train, y_train), shuffle=True,␣
 ↪batch_size=8)
model.lstm.dropout=0.2
```

```
torch.Size([1090, 250, 1]) torch.Size([1090, 250, 1])
torch.Size([410, 250, 1]) torch.Size([410, 250, 1])
```

1.3.5 LTSM Training

[ ]:
```
training_enabled = False
model_path = '/home/ubuntu/src/PML/LSTM-23-04-2023/model'
tag='24-04-2023-05-41'
CURRENT_MODEL_PATH = f'{model_path}/model-{tag}.pt'

if training_enabled:
    n_epochs = 5000
    for epoch in range(n_epochs):
        model.train()
        for X_batch, y_batch in loader:
            y_pred = model(X_batch)
            loss = gpu_loss_fn(y_pred, y_batch)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        # Validation
        if epoch % 100 != 0:
            continue
        model.eval()
        train_rmse=-1
        test_rmse=-1
```

```
        with torch.no_grad():
            y_pred = model(X_train)
            train_error = loss_fn(y_pred, y_train)
            train_rmse = np.sqrt(train_error.cpu().numpy())
            y_pred = model(X_test)
            test_error = loss_fn(y_pred, y_test)
            test_rmse = np.sqrt(test_error.cpu().numpy())
        print("Epoch %d: train RMSE %.4f, test RMSE %.4f" % (epoch, train_rmse,␣
 ↪test_rmse))

        # Save the trained model's parameters and architecture to a file
        torch.save(model.state_dict(), CURRENT_MODEL_PATH) # Commented out for␣
 ↪accidental overriding.
        # Optionally, save other necessary information such as hyperparameters
        with open(f'{model_path}/hyperparameters-{tag}.txt', 'w') as f:
            f.write(f'input_size: {1}\n')
            f.write(f'hidden_size: {52}\n')
            f.write(f'num_layers: {2}\n')
            f.write(f'output_size: {1}\n')
            f.write(f'batch_size: {10000}\n')
            f.write(f'num_epochs: {20000}\n')
else:
     # Load the saved model.
    loaded_model = LSTM_Model()
    use_live_model = False
    if use_live_model:
        loaded_model = model
    else:
        loaded_model.load_state_dict(torch.load(CURRENT_MODEL_PATH))
```

[ ]:
```
# Perform inference with LSTM Model
with torch.no_grad():  # Disable gradient computation
    y_train_pred = (loaded_model.cpu())(X_train.cpu())
    train_error = (loss_fn.cpu())(y_train_pred.cpu(), y_train.cpu())
    train_rmse = np.sqrt(train_error.cpu().numpy())
    print("Final train RMSE: %.4f" % train_rmse)

    y_test_pred = (loaded_model.cpu())(X_test.cpu())
    test_error = (loss_fn.cpu())(y_test_pred.cpu(), y_test.cpu())
    test_rmse = np.sqrt(test_error.cpu().numpy())
    print("Final test RMSE: %.4f" % test_rmse)
```

```
Final train RMSE: 1.0990
Final test RMSE: 18.1457
```

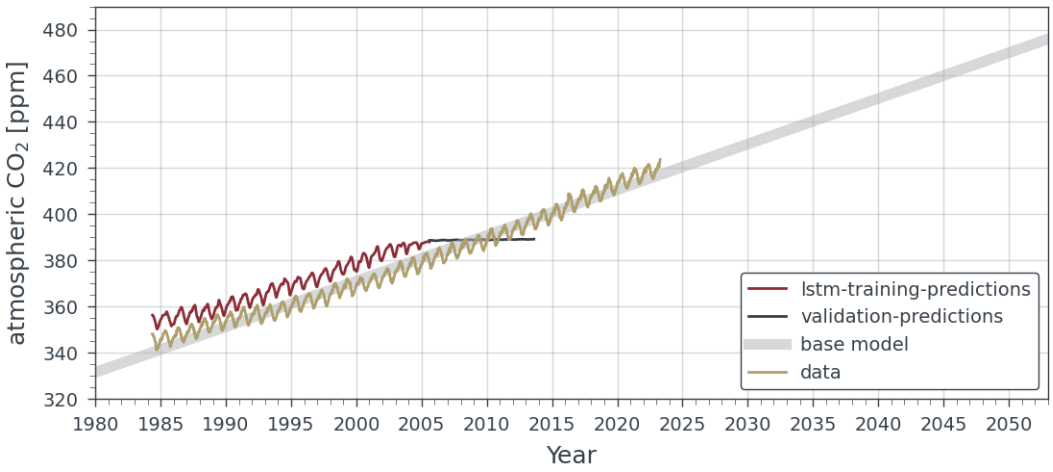
[ ]:
```
# Perform inference with LSTM Model
with torch.no_grad():  # Disable gradient computation
```

```
    y_train_pred = (loaded_model.cpu())(X_train.cpu())
    train_error = (loss_fn.cpu())(y_train_pred.cpu(), y_train.cpu())
    train_rmse = np.sqrt(train_error.cpu().numpy())
    print("Final train RMSE: %.4f" % train_rmse)

    y_test_pred = (loaded_model.cpu())(X_test.cpu())
    test_error = (loss_fn.cpu())(y_test_pred.cpu(), y_test.cpu())
    test_rmse = np.sqrt(test_error.cpu().numpy())
    print("Final test RMSE: %.4f" % test_rmse)
```

```
Final train RMSE: 1.0990
Final test RMSE: 18.1457
```

[ ]:
```
# Model is tested on the test set as follows.
with torch.no_grad():  # Disable gradient computation
    y_train_pred = (loaded_model.cpu())(X_train.cpu())
    train_error = (loss_fn.cpu())(y_train_pred.cpu(), y_train.cpu())
    train_rmse = np.sqrt(train_error.cpu().numpy())
    print("Final train RMSE: %.4f" % train_rmse)
```

[ ]:
```
fig, ax = plt.subplots()
# Add Training set data for comparision.

#plt.plot(X.to_numpy()[:train_size], timeseries[:train_size], label='data')
plt.plot(X.to_numpy()[0:y_train_pred.shape[0]], y_train_pred[:, -1,:].cpu().
 ↪numpy(),label='lstm-training-predictions')
plt.plot(X.to_numpy()[y_train_pred.shape[0]:y_train_pred.shape[0]+y_test_pred.
 ↪shape[0]], y_test_pred[:,-1,:].cpu().numpy(), label='validation-predictions')

# Test set data.
x = jnp.linspace(1980, 2053, 200)
ax.plot(x, w[0] + w[1] * x, color=rgb.tue_gray,
        lw=4, alpha=0.5, label="base model")
ax.plot(X, Y, label="data")

ax.set_xlabel("Year")
ax.set_ylabel("atmospheric CO$_2$ [ppm]")
ax.xaxis.set_major_locator(ticker.MultipleLocator(5))
ax.xaxis.set_minor_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(20))
ax.yaxis.set_minor_locator(ticker.MultipleLocator(5))
ax.set_xlim([1980, 2053])
ax.set_ylim([320, 490])
ax.grid(which="major", axis="both")
ax.legend(loc="lower right", framealpha=1, facecolor="w");
```



2 References

- MLP for time series forecasting: https://machinelearningmastery.com/how-to-develop-multilayer-perceptron-models-for-time-series-forecasting/
- LSTM for time series prediction: https://machinelearningmastery.com/lstm-for-time-series-prediction-in-pytorch/
- LSTM: https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html
- The Unreasonable Effectiveness of Recurrent Neural Networks: http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- Radial Basis Functions https://deepai.org/machine-learning-glossary-and-terms/radial-basis-function#:~:text=What%20is%20a%20Radial%20Basis,denoted%20C%2C%20called%20a%20center.