Exercise Sheet 7 (programming part)

In this homework, our goal is to try out recurrent neural network layers in PyTorch.

Part 1: Visualizing the data

Because gradient computation can be error-prone, we often rely on libraries that incorporate automatic differentiation. In this exercise, we make use of the PyTorch library. You are then asked to compute the error of the neural network within that framework, which will then be automatically differentiated.

In [ ]:
```
import torch
import torch.nn as nn
import matplotlib.pyplot as plt

import solution07
import utils07 as utils

# 1. Get the data and parameters
data = utils.getdata()

# 2. Visualize the time series
plt.plot(data.T)

input = torch.from_numpy(data[:, :-1])
target = torch.from_numpy(data[:, 1:])
```

Part 2: Implementing a LSTM Network (20 P)

Implement a two layer LSTM network with pytorch.nn.LSTMCell with 25 hidden neurons. At each prediction step use a linear projection layer to project the hidden representation back to the original data space.

In [ ]:
```
class LSTM_Network(nn.Module):
        def __init__(self):
                super().__init__()  # implement two LSTMCell layers  # and one linear projecti
on layer

        def forward(self, input, prediction_steps=0):
                # Compute the LSTM's predictions over the input
                # Compute 'prediction_steps' steps of predictions into the future
                # Return the concatenated outputs
                pass

lstm = None
# lstm = solution07.exercise1()
```

Part 3: Train the LSTM and Visualize It (10 P)

As a last exercise, we would like to make use of existing neural network objects of the PyTorch library. Here, most of the code is already implemented for you. You are only asked to find where the error gradient of the first weight parameter has been stored, and to print it.

In [ ]:
```
from torch import optim

lstm.double()
criterion = nn.MSELoss()
optimizer = optim.Adam(lstm.parameters())
num_steps = 100
for i in range(num_steps):
        #
        # implement a training step
        # zero the grads, pred, compute loss, backpropagate grad
        #
        if i % (num_steps//10) == 0:
                print(f'Training Progress: {int(i/num_steps*100)}%, Loss:', loss.item())

with torch.no_grad():
        future = 100
        pred = lstm(input, future=future)
        y = pred.detach().numpy()
```

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(figsize=(30, 10))
plt.title('Solid Lines are Training Data and Dashed Lines are Predictions into the Future', fo
ntsize=30)
plt.xlabel('x', fontsize=20)
plt.ylabel('y', fontsize=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)

def draw(yi, color):
        plt.plot(np.arange(input.size(1)), yi[:input.size(1)], color, linewidth=2.0)
        plt.plot(np.arange(input.size(1), input.size(1) + future), yi[input.size(1):], color +
':', linewidth=2.0)

draw(y[0], 'r')
draw(y[1], 'g')
```