```
import torch
import torch.nn
from torch import nn


def exercise2():
    class LSTMNetwork(nn.Module):
        def __init__(self):
            super().__init__()
            self.lstm1 = nn.LSTMCell(1, 51)
            self.lstm2 = nn.LSTMCell(51, 51)
            self.linear = nn.Linear(51, 1)

        def forward(self, input, future=0):
            outputs = []
            h_t = torch.zeros(input.size(0), 51, dtype=torch.double)
            c_t = torch.zeros(input.size(0), 51, dtype=torch.double)
            h_t2 = torch.zeros(input.size(0), 51, dtype=torch.double)
            c_t2 = torch.zeros(input.size(0), 51, dtype=torch.double)

            for input_t in input.split(1, dim=1):
                h_t, c_t = self.lstm1(input_t, (h_t, c_t))
                h_t2, c_t2 = self.lstm2(h_t, (h_t2, c_t2))
                output = self.linear(h_t2)
                outputs += [output]

            for i in range(future):  # if we should predict the future
                h_t, c_t = self.lstm1(output, (h_t, c_t))
                h_t2, c_t2 = self.lstm2(h_t, (h_t2, c_t2))
                output = self.linear(h_t2)
                outputs += [output]
            outputs = torch.cat(outputs, dim=1)
            return outputs

    lstm = LSTMNetwork()
    return lstm


def exercise3():
    '''
    optimizer.zero_grad()
    out = lstm(input)
    loss = criterion(out, target)
    loss.backward()
    optimizer.step()
    '''
```