```
# -*- coding: utf-8 -*-
__author__ = 'Gerald Schuller'
__copyright__ = 'G.S.'

"""
Simple program for using a recurrent neural network in pytorch as an IIR filter.
Needs to be executed py python3.
See: https://pytorch.org/docs/stable/nn.html
Input is an impulse, output a decaying tone, as in pyrecplay_2ndOrdIIRkeyboarddisplay.py
See also:
https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html
and in:
https://pytorch.org/docs/stable/nn.html?highlight=rnn#torch.nn.RNN
Output is the last state h, geht immer durch eine non-linearity.

Gerald Schuller, June 2020.
"""

import torch
import torch.nn as nn
import numpy as np
import scipy.signal
import soundfloat
import os
#import optimrandomdir_pytorch
#import optimrandomdir_pytorch_parallel as optimrandomdir_pytorch
import time
import matplotlib.pyplot as plt

infeatures= 1 # samples per input time step
hiddensize= 3 #number of hidden states, the memory elements
outputs=1 #1 samples per output time step
numlayers=1 #number of layers of the network

batch=1 #number of audio signals for training
Fs=8000 #sampling rate of the audio signal
seq_len=int(Fs*0.6) #0.6 second sound

# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device='cpu'
print("device=", device)

class RNNnet(nn.Module):
    def __init__(self, infeatures, hiddensize, outputs):
      super(RNNnet, self).__init__()
      # Define the model.
      self.rnn = nn.RNN(input_size=infeatures, hidden_size=hiddensize, num_layers=numlayers,
bias=False)
      #forward layer for output
      self.fo = nn.Linear(hiddensize, outputs, bias=False)



    def forward(self, x):
      h_0 = torch.zeros(numlayers, batch, hiddensize).to(device)
      out, hn = self.rnn(x, h_0)
      #Output is simply the hidden state of the last layer (if more than 1 layer)
      out = self.fo(out) #e.g. used to just keep first output
      return out

def signal2pytorch(x):
   #Function to convert a signal vector x, like a mono audio signal, into a 3-d Tensor that
Pytorch expects,
   #https://pytorch.org/docs/stable/nn.html
   #Argument x: a 1-d signal as numpy array
   #output: 3-d Pytorch Tensor.
   #for RNN Input: (siglen,batch,features)
   X = np.expand_dims(x, axis=-1)  #add batch  dimension (here only 1 )
```

```
    X = np.expand_dims(X, axis=-1)  #add features dimension (here only 1 )
    X=torch.from_numpy(X)
    X=X.type(torch.Tensor)
    return X

#-------------------------------------------------------------------------
#IIR Filter coefficients:
omega=2*np.pi*440.0 /Fs; #normalized frequency
print("frequency= ", omega*Fs/(2*np.pi))
#Q is the quality factor (1 decays forever)
Q=0.995 #determines speed of decay, the closer to 1 the longer
#for Q=0.9995 and 8 kHz sampling rate Adam is not converging, but at 2 kHz sampling rate
#Means: longer decay, longer memory -> more difficult gradient computation
#k1 and k2 are the resulting denominator coefficients:
#See also Grundlagen der Videotechnik, WS13/14, number 12:
k1=2*Q*np.cos(omega)
k2=-pow(Q,2);
print("k1, k2=", k1,k2)

ximp=np.zeros(seq_len) #make impulse with a desired sequence length
ximp[0]=0.1 #impulse at the beginning
filtered=scipy.signal.lfilter([1], np.array([1, -k1,-k2]), ximp)
print("The impulse response sound from the I.I.R. filter")
os.system('espeak -s 120 "The impulse response sound from the I.I.R. filter"')
soundfloat.sound(filtered, Fs)

"""
RNN as IIR Filter: hidden states h as delay chain
Weights for updating h for shifting the delay line:
s=array([[0,0,0],[1,0,0],[0,1,0]])
In [4]: s
array([[0, 0, 0],
       [1, 0, 0],
       [0, 1, 0]])

Application of IIR coefficients a(1)=2, a(2)=3:
a=array([[1,2,3],[0,1,0],[0,0,1]])

In [2]: a
Out[2]:
array([[1, 2, 3],
       [0, 1, 0],
       [0, 0, 1]])
First apply shift operator s and then coefficents, h multiplied from right:
dot(a,s)
array([[2, 3, 0],
       [1, 0, 0],
       [0, 1, 0]])
this is w_hh.
"""
#-----------------------------------------------------------------------------------
#Recurrent Neural Network model:
#hidden state update formula:
#ht=tanh(wih xt+bih+whh h(tâˆ’1)+bhh)

#rnn = nn.RNN(input_size=infeatures, hidden_size=hiddensize, num_layers=numlayers, bias=False)

rnn = RNNnet(infeatures, hiddensize, outputs).to(device)

#ww = rnn.state_dict()   #read obtained weights
#print("weights =", ww)

#Assign the weights such that the RNN corresponds to the IIR filter, except for the tanh
activation function:
#Shifted diagonal matrix and IIR Filter coefficients:
rnn.state_dict()['rnn.weight_hh_l0'].data.copy_(torch.diag(torch.ones(hiddensize-1),diagonal=-1))
rnn.state_dict()['rnn.weight_hh_l0'][0,:].data.copy_(torch.tensor([ k1, k2,0.0]))
#Vector with a "1" for input x:
rnn.state_dict()['rnn.weight_ih_l0'].data.copy_(torch.zeros((hiddensize,1)))
rnn.state_dict()['rnn.weight_ih_l0'][0,0].data.copy_(torch.tensor(1.0))
#Weight for linear output layer to select h[0]:
rnn.state_dict()['fo.weight'][0,:].data.copy_(torch.tensor([1.0, 0.0 ,0.0]))

ww = rnn.state_dict()   #read obtained weights
print("weights =", ww)

#the input, converted from the Numpy array:
inputsig=signal2pytorch(ximp)

print("inputsig.type()=", inputsig.type())
print("inputsig=", inputsig)

#Run Recurrent Neural Network:
outsig= rnn(inputsig)

#output of shape (seq_len, batch, hidden_size)
#print("output=", output)
#outsig=output[:,0,0] #take first output or "feature" h(0) of hidden layer size, corresponging to
the first delay element of the IIR filter.

outsig=outsig.detach()
outsig=np.array(outsig) #turn into numpy array
print("outsig[0:6]=", outsig[0:6])
print("comparison: filtered[0:6]=",filtered[0:6])

outsig=outsig[:,0,0]

print("The sound from the recurrent neural network")
os.system('espeak -s 120 "The impulse response sound from the calculated recurrent neural
network"')
soundfloat.sound(outsig, Fs)

plt.plot(filtered)
plt.plot(outsig)
plt.legend(('IIR Filter impulse resp., (target)','RNN Output'))
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title("Impulse response from the IIR and the computed RNN")
plt.show()


#----------------------------------------------------------------------------------------
#Version with obtaining the RNN weights from training,
#input is the pulse, with the 0.1 in the beginning, target is the output of the IIR filter:

target= signal2pytorch(filtered)
print("target.type()=", target.type())
print("target=", target)

#random starting point as initialization:
#rnn.state_dict()['rnn.weight_hh_l0'].data.copy_(torch.randn(hiddensize,hiddensize))
rnn.state_dict()['rnn.weight_hh_l0'].data.copy_(torch.zeros(hiddensize,hiddensize))
#rnn.state_dict()['rnn.weight_ih_l0'].data.copy_(torch.randn((hiddensize,1)))
rnn.state_dict()['rnn.weight_ih_l0'].data.copy_(torch.zeros((hiddensize,1)))
rnn.state_dict()['fo.weight'][0,:].data.copy_(torch.tensor([ 1.0, 0.0 ,0.0]))
#rnn.state_dict()['fo.weight'][0,:].data.copy_(torch.zeros(hiddensize))
#rnn.state_dict()['fo.weight'][0,:].data.copy_(torch.randn(hiddensize)))

#rnn.__init__(infeatures, hiddensize)
rnn = RNNnet(infeatures, hiddensize, outputs).to(device)

ww = rnn.state_dict()   #read current weights
print("weights =", ww)
print('Total number of parameters: %i' % (sum(p.numel() for p in rnn.parameters() if
p.requires_grad)))

loss_fn = nn.MSELoss(size_average=False)
#learning_rate = 1e-3
```

```
optimizer = torch.optim.Adam(rnn.parameters())#, lr=learning_rate)

Ypred=rnn(inputsig)
print("starting loss:", loss_fn(Ypred, target).item())
#print("Start optimization")
print("Starting training the neural network")
os.system('espeak -s 120 "Starting training the neural network"')

randdir=False #True for optimization of random directions
#--Start optimization of random directions------------------------
if randdir == True:
   #optimization of weights using method of random directions:
   starttime=time.time()

   optimrandomdir_pytorch.optimizer(rnn, loss_fn, inputsig, target, iterations=1500,
startingscale=0.1, endscale=0.0)

   endtime=time.time()
   print("Duration of optimization:", endtime-starttime)
   #--End optimization of random directions------------------------

else:
   starttime=time.time()
   for epoch in range(500):
       Ypred=rnn(inputsig)
       loss=loss_fn(Ypred, target)
       if epoch%1==0:
          print(epoch, loss.item())
       optimizer.zero_grad()
       loss.backward()
       optimizer.step()
   endtime=time.time()
   print("Duration of optimization:", endtime-starttime)

outsig= rnn(inputsig)
outsig=outsig.detach()
outsig=np.array(outsig) #turn into numpy array
outsig=outsig[:,0,0]

plt.plot(filtered)
plt.plot(outsig)
plt.legend(('Audio target','RNN Output'))
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title("Impulse Response from IIR and the trained RNN")
plt.show()

os.system('espeak -s 120 "The impulse response sound from the trained recurrent neural network"')
soundfloat.sound(outsig, Fs)

ww = rnn.state_dict()   #read obtained weights
print("weights =", ww)
```