

Neurális hálózatok házi feladat beszámoló

Pintér Bálint (I6QS0K), Szilágyi Gábor (NOMK01)

Koncentrált paraméterű RF szűrő optimalizációja aktív tanulással

1. Bevezetés

1.1. Aktív tanulás

A legtöbb neurális hálózatokat használó megoldás olyan problémára irányul, ahol sok rendelkezésre álló adat alapján kell a hálót betanítani egy feladat elvégzésére. Az a legtöbb esetben teljesül, hogy még több tanító adat felhasználásával jobb hálót lehetne tanítani, de ennek az extrém esetére tud megoldást nyújtani az aktív tanulás. A nem adathiányos problémáknál a rendelkezésre álló, címkézett adatpontok nagy részét felhasználva szokás tanítani a hálót, majd a fennmaradó adatpontokon ellenőrizni a háló teljesítőképességét olyan esetekre, amikkel nem találkozott a tanulás során. Az aktív tanulás folyamata ettől merőben eltér.

Aktív tanulás kiindulási helyzete, hogy nagyon sok címkézetlen adat áll rendelkezésre, de az egyes adatpontok címkézése rendkívül költséges. A címkézés költsége miatt végeredményben az a cél, hogy azt minél kevesebbszer kelljen elvégezni a tanulás során. A tanulási folyamat közben az eddig megkapott kevés címkézett adatpont alapján a háló jelöl ki következőnek címkézésre azt, amelyik várhatóan a leghasznosabb lesz számára. A hasznosság becslésére több megközelítés is létezik, erre a későbbiekben visszatérünk.

Az aktív tanulás egyik esete a Bayes-optimalizáció. Itt nem egy osztályozót tanítunk minél kevesebb címkézett adat alapján, hanem egy „fekete doboz” függvény maximumát keressük a függvény minél kevesebb kiértékelése mellett. Ez a különbség már befolyásolni fogja a következőnek megcímkézendő adat választását, ami ebben az esetben a következő paraméterértékek megválasztását jelenti, ahol kiértékeljük a függvényt.

1.2. Az optimalizálandó probléma

Az optimalizáció céljának

1.3. A felhasznált könyvtár

2. A probléma átalakítása

2.1. Impedanciák és admittanciák

2.2. Láncparaméterek

2.3. Szórási paraméterek

3. A célfüggvény

3.1. Az első verzió és a problémái

3.2. A paraméterek logaritmizálása

3.3. A függvényérték logaritmizálása

A. Az általunk írt specifikáció osztály

```
1 import numpy as np
2 import math
3 import matplotlib.pyplot as plt
4 from matplotlib.patches import Rectangle
5 from matplotlib.backends.backend_pdf import PdfPages
6
7 pi = math.pi
8 inf = math.inf
9 nan = math.nan
10
11 #impedance of the series LC subcircuit
12 def Impedance(f, lnC, lnL):
13     match [math.isnan(lnC), math.isnan(lnL)]:
14         case [True, True]:
15             return 0.0
16         case [True, False]:
17             L = math.exp(lnL)
18             return f*1j*2*pi*L
19         case [False, True]:
20             C = math.exp(lnC)
21             return -1j*1/(f*2*pi*C)
22         case _:
23             C = math.exp(lnC)
24             L = math.exp(lnL)
25             ZC = -1j*1/(f*2*pi*C)
26             ZL = 0+f*1j*2*pi*L
27             return ZL*ZC/(ZL+ZC)
28
29 def Admittance(f, lnC, lnL):
30     match [math.isnan(lnC), math.isnan(lnL)]:
31         case [True, True]:
32             return 0.0
33         case [True, False]:
34             L = math.exp(lnL)
35             return 1/(f*1j*2*pi*L)
36         case [False, True]:
37             C = math.exp(lnC)
38             return f*2*pi*C*1j
39         case _:
40             C = math.exp(lnC)
41             L = math.exp(lnL)
42             ZC = -1j*1/(f*2*pi*C)
43             ZL = 0+f*1j*2*pi*L
44             return (ZL+ZC)/ZL*ZC
45
46 class Spec:
47     def __init__(self, starts, ends, limits, directions, margin, n):
48         # starts of pass- or stopbands in Hz
49         self.starts = starts
50         # ends of start- or stopbands in Hz
51         self.ends = ends
52         # pass- or stopband threshold values
53         self.limits = limits
54         # pass- or stopbands? possible values: "pass" or "stop" strings
55         self.directions = directions
56         # number of frequency points
57         self.n = n
58         # margin to still punish solution that only barely satisfies the specification
59         # on stopbands, with limit l and margin m, the margin range is from l*(1-m) to l
60         # on passbands, with limit l and margin m, the margin range is from l to l*(1+m)
61         self.margin = margin
62
63     #def cost(self, types, values):
64     def cost(self, plot=False, par1C=nan, par1L=nan, ser1C=nan, ser1L=nan, par2C=nan, par2L=
65         nan, ser2C=nan, ser2L=nan, par3C=nan, par3L=nan, ser3C=nan, ser3L=nan):
66         """Parameters define a ladder structure,
67         where each series or parallel element consists of
68         two discrete, ideal L or C components in parallel
69         with each other. Default values define a perfect
70         all-pass filter. In the function parameters, "parXY" means
```

```

70     the value of the Y-th sub-component of the X-th parallel
71     element. Similarly, "serXY" means the value of the Y-th
72     sub-component of the X-th series element. The ladder starts
73     with a parallel element."""
74     values = [par1C, par1L, ser1C, ser1L, par2C, par2L, ser2C, ser2L, par3C, par3L, ser3C,
ser3L]
75     nval = len(values)
76     nladder = int(nval/4)
77     minf = (min(self.starts))
78     maxf = (max(self.ends))
79     # frequency axis sample points (log spacing)
80     faxis = []
81     # no. of frequency sample points
82     #n = self.n
83     factor = (maxf/minf)**((1/self.n))
84     for i in range(self.n+1):
85         faxis.append(minf*factor**(i))
86     # reference impedance in Ohm, on both ports
87     Z0 = 50
88     Y0 = 1/Z0
89     # array of overall S21 values at the frequency sample points
90     S21 = []
91     #process 1 parallel and 1 series element:
92     for f in faxis:
93         # ABCD parameter matrix of the whole system
94         ABCD = np.matrix([[1,0],[0,1]])
95         for l in range(nladder):
96             # admittance of the two parallel components together from the current step of
the ladder
97             Y = Admittance(f, values[4*l+0], values[4*l+1])
98             mPar = np.matrix([[1, 0],[Y, 1]])
99             Z = Impedance(f, values[4*l+2], values[4*l+3])
100             mSer = np.matrix([[1, Z],[0, 1]])
101             ABCD = ABCD*mPar*mSer
102             #  $2/(A+B/Z_0+C*Z_0+D)$ 
103             A = ABCD.item(0,0)
104             B = ABCD.item(0,1)
105             C = ABCD.item(1,0)
106             D = ABCD.item(1,1)
107             S21.append(abs(2/(A+B/Z0+C*Z0+D)))
108     # natural log of margin+1
109     lnmargin=math.log(self.margin+1)
110     if(plot):
111         # green: passband; red: stopband; orange: ok, but close to not ok, still punished
112         fig, ax = plt.subplots()
113         minS21 = min(S21)
114         for i in range(len(self.starts)):
115             if(self.directions[i] == "pass"):
116                 # region forbidden by the original specification
117                 ax.add_patch(Rectangle((self.starts[i], minS21),
self.ends[i]-self.starts[i],
self.limits[i]-minS21,
facecolor='#00aa00'))
118                 # region close to original limit, but satisfying it, additional penalty
region
119                 ax.add_patch(Rectangle((self.starts[i], self.limits[i]),
(self.ends[i]-self.starts[i]),
self.limits[i]*self.margin,
facecolor='orange'))
120             else: # "stop"
121                 # region forbidden by the original specification
122                 ax.add_patch(Rectangle((self.starts[i], self.limits[i]),
self.ends[i]-self.starts[i],
1.0-self.limits[i],
facecolor='#aa0000'))
123                 # region close to original limit, but satisfying it, additional penalty
region
124                 ax.add_patch(Rectangle((self.starts[i], self.limits[i]*(1/(1+self.margin))
),
(self.ends[i]-self.starts[i]),
self.limits[i]*(1-1/(1+self.margin)),
facecolor='orange'))
125         ax.loglog(faxis, S21, 'k-')

```

```

138         plt.ylabel(r'$|S_{21}|$')
139         plt.xlabel(r'$f$')
140         plt.savefig("plot.pdf", dpi=120, format='pdf', bbox_inches='tight')
141         plt.show()
142     cost = 0
143     # number of freq points in the regions where the S21 is specified
144     ncost = 0
145     for i in range(len(self.starts)):
146         for j in range(len(faxis)):
147             if faxis[j]>self.starts[i] and faxis[j]<self.ends[i]:
148                 ncost = ncost + 1
149                 lnlimit = math.log(self.limits[i])
150                 lnS21 = math.log(S21[j])
151                 if self.directions[i] == "pass":
152                     cost += max(0, min(-lnS21, lnlimit+lnmargin-lnS21))
153                 else: # "stop"
154                     cost += max(0, lnS21-lnlimit+lnmargin)
155     # negative of average cost, for function maximizing
156     return -cost/ncost

```