

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Learning a Linear Function via Perceptron and Backpropagation for Neural Network Learning**Objective:**

The main goal of part A is to implement a perceptron model that learns a Boolean function (e.g. AND, OR, NAND, NOR) and the objective of part B is to implement the in-class exercise/tutorial on multi-layer perceptron learning.

Background:

For part A, we have written a simple program that uses a single perceptron which uses a Sum of Products (S) operation together with a “step” function (y) for activation:

$$S = \sum_{i=1}^d w_i x_i + w_{bias} \quad y = \begin{cases} 1, & \text{if } S > 0 \\ 0, & \text{otherwise} \end{cases}$$

where w are parameters (or weights), x are the inputs and d is the dimension.

There are two modes in our program such as “Classification mode” and “learning mode”. In “classification” mode, it accepts two inputs (0|1) and produces an Output (0|1) that corresponds to that function. In learning mode, it iteratively feeds forward two Inputs, produces an Output, minimizes Error, and uses the below Delta rule to update the three weights which were chosen randomly first with a fixed seed of 1:

$$\Delta w_{i,j} = \eta(t_j - y_j)x_i$$

where t is the target and η is the learning rate.

After running through a few Epochs, it produces correct Outputs for all four Input combinations. As a result, it has learned the function. The below output is for AND function:

```
***** AND *****
Initial Weights: 0.13 [0.85 0.76]
*****

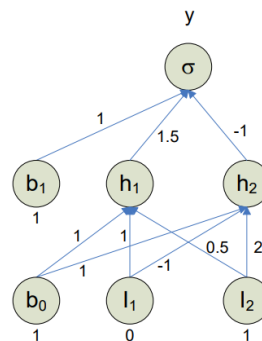
Epoch #1 | Epoch #2 |
-----|-----|
inputs: [0, 0] | inputs: [0, 0] | Solution took 2 epochs.
weights: 0.13 [0.85 0.76] | weights: -0.87 [0.85 0.26] | Final weights: -0.87 [0.85 0.26]
y=1 t=0 ==> incorrect | y=0 t=0 ==> correct |
inputs: [0, 1] | inputs: [0, 1] | ***** Classify *****
weights: -0.37 [0.85 0.76] | weights: -0.87 [0.85 0.26] | inputs: [0, 1]
y=1 t=0 ==> incorrect | y=0 t=0 ==> correct | weights: -0.87 [0.85 0.26]
inputs: [1, 0] | inputs: [1, 0] | Prediction: 0
weights: -0.87 [0.85 0.26] | weights: -0.87 [0.85 0.26] |
y=0 t=0 ==> correct | y=0 t=0 ==> correct |
inputs: [1, 1] | inputs: [1, 1] |
weights: -0.87 [0.85 0.26] | weights: -0.87 [0.85 0.26] |
y=1 t=1 ==> correct | y=1 t=1 ==> correct |

*****
```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

For part B, we have written a program that implements a multi-layer neural network and demonstrates iterative learning. There was an additional hidden layer between the input and output layers in the multi-layer perceptron learning tutorial.

Below is a snapshot of a neural network during training. There are two input units, two hidden layer perceptrons, and a single output unit. Input I1 has a value of 0; input I2 has a value of 1; all bias has value 1. Edges are labeled with their corresponding weights. Learning factor $\eta = 0.5$. Target value is 1.



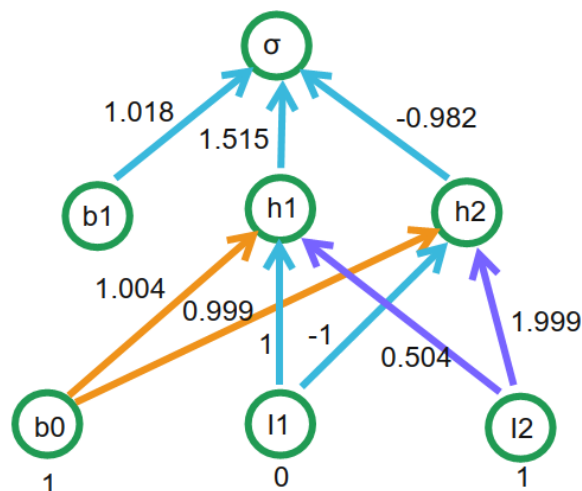
As a form of Supervised Learning, the training process consists of repeatedly presenting labeled examples to the system and iteratively adjusting parameters (weights) until the desired outcome is obtained:

$$\text{WeightUpdate} = \text{LearningRate} \cdot (\text{TargetOutput} - \text{ActualOutput}) \cdot \text{Input}$$

In a multilayer network, there is no target value for perceptrons in the hidden layer; hence their error cannot be directly calculated. Therefore, error is propagated backwards from the output layer.

The below snapshot is the neural network after updating the weights:

**Final network with the updated weights
(Manually drawn using the final weights)**



Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Visualization: [Display the Output]

Output of Part A:

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```

1 C:\Users\shuvo\AppData\Local\Programs\Python\Python39\python.exe "C:\Program Files\JetBrains\PyCharm 2021.3\plugins\
  python\helpers\pydev\pydevconsole.py" --mode=client --port=54776
2
3 import sys; print('Python %s on %s' % (sys.version, sys.platform))
4 sys.path.extend(['C:\\Users\\shuvo\\Workspace\\PyCharm\\CIS678', 'C:/Users/shuvo/Workspace/PyCharm/CIS678'])
5
6 PyDev console: starting.
7
8 Python 3.9.12 (tags/v3.9.12:b28265d, Mar 23 2022, 23:52:46) [MSC v.1929 64 bit (AMD64)] on win32
9 >>> runfile('C:/Users/shuvo/Workspace/PyCharm/CIS678/project4/project.py', wdir='C:/Users/shuvo/Workspace/PyCharm/CIS678
  /project4')
10
11
12 ***** AND *****
13
14
15 Initial Weights: 0.13 [0.85 0.76]
16
17 *****
18
19 Epoch #1
20
21 inputs: [0, 0]
22 weights: 0.13 [0.85 0.76]
23 y=1 t=0 ==> incorrect
24
25 inputs: [0, 1]
26 weights: -0.37 [0.85 0.76]
27 y=1 t=0 ==> incorrect
28
29 inputs: [1, 0]
30 weights: -0.87 [0.85 0.26]
31 y=0 t=0 ==> correct
32
33 inputs: [1, 1]
34 weights: -0.87 [0.85 0.26]
35 y=1 t=1 ==> correct
36
37 *****
38
39 Epoch #2
40
41 inputs: [0, 0]
42 weights: -0.87 [0.85 0.26]
43 y=0 t=0 ==> correct
44
45 inputs: [0, 1]
46 weights: -0.87 [0.85 0.26]
47 y=0 t=0 ==> correct
48
49 inputs: [1, 0]
50 weights: -0.87 [0.85 0.26]
51 y=0 t=0 ==> correct
52
53 inputs: [1, 1]
54 weights: -0.87 [0.85 0.26]
55 y=1 t=1 ==> correct
56
57
58 Solution took 2 epochs.
59 Final weights: -0.87 [0.85 0.26]
60
61 ***** Classify *****
62
63 inputs: [0, 1]
64 weights: -0.87 [0.85 0.26]
65 Prediction: 0
66
67
68 ***** OR *****
69
70
71 Initial Weights: 0.13 [0.85 0.76]
72
73 *****
74
75 Epoch #1
76
77 inputs: [0, 0]
78 weights: 0.13 [0.85 0.76]

```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```
79 y=1 t=0 ==> incorrect
80
81 inputs: [0, 1]
82 weights: -0.37 [0.85 0.76]
83 y=1 t=1 ==> correct
84
85 inputs: [1, 0]
86 weights: -0.37 [0.85 0.76]
87 y=1 t=1 ==> correct
88
89 inputs: [1, 1]
90 weights: -0.37 [0.85 0.76]
91 y=1 t=1 ==> correct
92
93 *****
94
95 Epoch #2
96
97 inputs: [0, 0]
98 weights: -0.37 [0.85 0.76]
99 y=0 t=0 ==> correct
100
101 inputs: [0, 1]
102 weights: -0.37 [0.85 0.76]
103 y=1 t=1 ==> correct
104
105 inputs: [1, 0]
106 weights: -0.37 [0.85 0.76]
107 y=1 t=1 ==> correct
108
109 inputs: [1, 1]
110 weights: -0.37 [0.85 0.76]
111 y=1 t=1 ==> correct
112
113
114 Solution took 2 epochs.
115 Final weights: -0.37 [0.85 0.76]
116
117 ***** Classify *****
118
119 inputs: [0, 1]
120 weights: -0.37 [0.85 0.76]
121 Prediction: 1
122
123
124 ***** NAND *****
125
126
127 Initial Weights: 0.13 [0.85 0.76]
128
129 *****
130
131 Epoch #1
132
133 inputs: [0, 0]
134 weights: 0.13 [0.85 0.76]
135 y=1 t=1 ==> correct
136
137 inputs: [0, 1]
138 weights: 0.13 [0.85 0.76]
139 y=1 t=1 ==> correct
140
141 inputs: [1, 0]
142 weights: 0.13 [0.85 0.76]
143 y=1 t=1 ==> correct
144
145 inputs: [1, 1]
146 weights: 0.13 [0.85 0.76]
147 y=1 t=0 ==> incorrect
148
149 *****
150
151 Epoch #2
152
153 inputs: [0, 0]
154 weights: -0.37 [0.35 0.26]
155 y=0 t=1 ==> incorrect
156
157 inputs: [0, 1]
158 weights: 0.13 [0.35 0.26]
```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```
159 y=1 t=1 ==> correct
160
161 inputs: [1, 0]
162 weights: 0.13 [0.35 0.26]
163 y=1 t=1 ==> correct
164
165 inputs: [1, 1]
166 weights: 0.13 [0.35 0.26]
167 y=1 t=0 ==> incorrect
168
169 *****
170
171 Epoch #3
172
173 inputs: [0, 0]
174 weights: -0.37 [-0.15 -0.24]
175 y=0 t=1 ==> incorrect
176
177 inputs: [0, 1]
178 weights: 0.13 [-0.15 -0.24]
179 y=0 t=1 ==> incorrect
180
181 inputs: [1, 0]
182 weights: 0.63 [-0.15 0.26]
183 y=1 t=1 ==> correct
184
185 inputs: [1, 1]
186 weights: 0.63 [-0.15 0.26]
187 y=1 t=0 ==> incorrect
188
189 *****
190
191 Epoch #4
192
193 inputs: [0, 0]
194 weights: 0.13 [-0.65 -0.24]
195 y=1 t=1 ==> correct
196
197 inputs: [0, 1]
198 weights: 0.13 [-0.65 -0.24]
199 y=0 t=1 ==> incorrect
200
201 inputs: [1, 0]
202 weights: 0.63 [-0.65 0.26]
203 y=0 t=1 ==> incorrect
204
205 inputs: [1, 1]
206 weights: 1.13 [-0.15 0.26]
207 y=1 t=0 ==> incorrect
208
209 *****
210
211 Epoch #5
212
213 inputs: [0, 0]
214 weights: 0.6299999999999999 [-0.65 -0.24]
215 y=1 t=1 ==> correct
216
217 inputs: [0, 1]
218 weights: 0.6299999999999999 [-0.65 -0.24]
219 y=1 t=1 ==> correct
220
221 inputs: [1, 0]
222 weights: 0.6299999999999999 [-0.65 -0.24]
223 y=0 t=1 ==> incorrect
224
225 inputs: [1, 1]
226 weights: 1.13 [-0.15 -0.24]
227 y=1 t=0 ==> incorrect
228
229 *****
230
231 Epoch #6
232
233 inputs: [0, 0]
234 weights: 0.6299999999999999 [-0.65 -0.74]
235 y=1 t=1 ==> correct
236
237 inputs: [0, 1]
238 weights: 0.6299999999999999 [-0.65 -0.74]
```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```
239 y=0 t=1 ==> incorrect
240
241 inputs: [1, 0]
242 weights: 1.13 [-0.65 -0.24]
243 y=1 t=1 ==> correct
244
245 inputs: [1, 1]
246 weights: 1.13 [-0.65 -0.24]
247 y=1 t=0 ==> incorrect
248
249 *****
250
251 Epoch #7
252
253 inputs: [0, 0]
254 weights: 0.6299999999999999 [-1.15 -0.74]
255 y=1 t=1 ==> correct
256
257 inputs: [0, 1]
258 weights: 0.6299999999999999 [-1.15 -0.74]
259 y=0 t=1 ==> incorrect
260
261 inputs: [1, 0]
262 weights: 1.13 [-1.15 -0.24]
263 y=0 t=1 ==> incorrect
264
265 inputs: [1, 1]
266 weights: 1.63 [-0.65 -0.24]
267 y=1 t=0 ==> incorrect
268
269 *****
270
271 Epoch #8
272
273 inputs: [0, 0]
274 weights: 1.13 [-1.15 -0.74]
275 y=1 t=1 ==> correct
276
277 inputs: [0, 1]
278 weights: 1.13 [-1.15 -0.74]
279 y=1 t=1 ==> correct
280
281 inputs: [1, 0]
282 weights: 1.13 [-1.15 -0.74]
283 y=0 t=1 ==> incorrect
284
285 inputs: [1, 1]
286 weights: 1.63 [-0.65 -0.74]
287 y=1 t=0 ==> incorrect
288
289 *****
290
291 Epoch #9
292
293 inputs: [0, 0]
294 weights: 1.13 [-1.15 -1.24]
295 y=1 t=1 ==> correct
296
297 inputs: [0, 1]
298 weights: 1.13 [-1.15 -1.24]
299 y=0 t=1 ==> incorrect
300
301 inputs: [1, 0]
302 weights: 1.63 [-1.15 -0.74]
303 y=1 t=1 ==> correct
304
305 inputs: [1, 1]
306 weights: 1.63 [-1.15 -0.74]
307 y=0 t=0 ==> correct
308
309 *****
310
311 Epoch #10
312
313 inputs: [0, 0]
314 weights: 1.63 [-1.15 -0.74]
315 y=1 t=1 ==> correct
316
317 inputs: [0, 1]
318 weights: 1.63 [-1.15 -0.74]
```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```
319 y=1 t=1 ==> correct
320
321 inputs: [1, 0]
322 weights: 1.63 [-1.15 -0.74]
323 y=1 t=1 ==> correct
324
325 inputs: [1, 1]
326 weights: 1.63 [-1.15 -0.74]
327 y=0 t=0 ==> correct
328
329
330 Solution took 10 epochs.
331 Final weights: 1.63 [-1.15 -0.74]
332
333 ***** Classify *****
334
335 inputs: [0, 1]
336 weights: 1.63 [-1.15 -0.74]
337 Prediction: 1
338
339
340 ***** NOR *****
341
342
343 Initial Weights: 0.13 [0.85 0.76]
344
345 *****
346
347 Epoch #1
348
349 inputs: [0, 0]
350 weights: 0.13 [0.85 0.76]
351 y=1 t=1 ==> correct
352
353 inputs: [0, 1]
354 weights: 0.13 [0.85 0.76]
355 y=1 t=0 ==> incorrect
356
357 inputs: [1, 0]
358 weights: -0.37 [0.85 0.26]
359 y=1 t=0 ==> incorrect
360
361 inputs: [1, 1]
362 weights: -0.87 [0.35 0.26]
363 y=0 t=0 ==> correct
364
365 *****
366
367 Epoch #2
368
369 inputs: [0, 0]
370 weights: -0.87 [0.35 0.26]
371 y=0 t=1 ==> incorrect
372
373 inputs: [0, 1]
374 weights: -0.37 [0.35 0.26]
375 y=0 t=0 ==> correct
376
377 inputs: [1, 0]
378 weights: -0.37 [0.35 0.26]
379 y=0 t=0 ==> correct
380
381 inputs: [1, 1]
382 weights: -0.37 [0.35 0.26]
383 y=1 t=0 ==> incorrect
384
385 *****
386
387 Epoch #3
388
389 inputs: [0, 0]
390 weights: -0.87 [-0.15 -0.24]
391 y=0 t=1 ==> incorrect
392
393 inputs: [0, 1]
394 weights: -0.37 [-0.15 -0.24]
395 y=0 t=0 ==> correct
396
397 inputs: [1, 0]
398 weights: -0.37 [-0.15 -0.24]
```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```
399 y=0 t=0 ==> correct
400
401 inputs: [1, 1]
402 weights: -0.37 [-0.15 -0.24]
403 y=0 t=0 ==> correct
404
405 *****
406
407 Epoch #4
408
409 inputs: [0, 0]
410 weights: -0.37 [-0.15 -0.24]
411 y=0 t=1 ==> incorrect
412
413 inputs: [0, 1]
414 weights: 0.13 [-0.15 -0.24]
415 y=0 t=0 ==> correct
416
417 inputs: [1, 0]
418 weights: 0.13 [-0.15 -0.24]
419 y=0 t=0 ==> correct
420
421 inputs: [1, 1]
422 weights: 0.13 [-0.15 -0.24]
423 y=0 t=0 ==> correct
424
425 *****
426
427 Epoch #5
428
429 inputs: [0, 0]
430 weights: 0.13 [-0.15 -0.24]
431 y=1 t=1 ==> correct
432
433 inputs: [0, 1]
434 weights: 0.13 [-0.15 -0.24]
435 y=0 t=0 ==> correct
436
437 inputs: [1, 0]
438 weights: 0.13 [-0.15 -0.24]
439 y=0 t=0 ==> correct
440
441 inputs: [1, 1]
442 weights: 0.13 [-0.15 -0.24]
443 y=0 t=0 ==> correct
444
445
446 Solution took 5 epochs.
447 Final weights: 0.13 [-0.15 -0.24]
448
449 ***** Classify *****
450
451 inputs: [0, 1]
452 weights: 0.13 [-0.15 -0.24]
453 Prediction: 0
454
455 Process finished with exit code 0
456
```


Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Output of Part B:

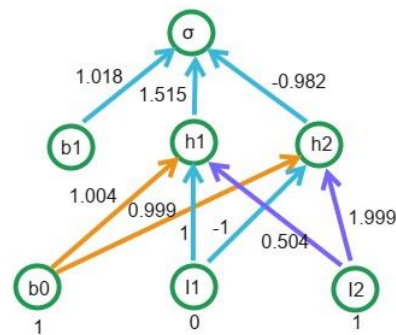
Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```

1 C:\Users\shuvo\AppData\Local\Programs\Python\Python39\python.exe "C:\Program Files\JetBrains\PyCharm 2021.3\plugins\
python\helpers\pydev\pydevconsole.py" --mode=client --port=54951
2
3 import sys; print('Python %s on %s' % (sys.version, sys.platform))
4 sys.path.extend(['C:\Users\shuvo\Workspace\PyCharm\CIS678', 'C:/Users/shuvo/Workspace/PyCharm/CIS678'])
5
6 PyDev console: starting.
7
8 Python 3.9.12 (tags/v3.9.12:b28265d, Mar 23 2022, 23:52:46) [MSC v.1929 64 bit (AMD64)] on win32
9 >>> runfile('C:/Users/shuvo/Workspace/PyCharm/CIS678/project4/neural_net_sample.py', wdir='C:/Users/shuvo/Workspace/
PyCharm/CIS678/project4')
10 Step 1: Feed the Inputs forward
11
12 Sigma(h1) = { (1.0+1)+(1.0+0)+(0.5+1) } = 1.5    h1 = 0.818
13 Sigma(h2) = { (1.0+1)+(-1.0+0)+(2.0+1) } = 3.0    h2 = 0.953
14 Sigma(y) = { (1.0+1)+(1.5+0.818)+(-1.0+0.953) } = 1.274    y = 0.781
15 Total error in network E = (0.5x(1-0.781)^2) = 0.024
16
17
18 Step 2: Backpropagate the errors
19
20 Error(y) = 0.781 x ( 1 - 0.781 ) x ( 1 - 0.781 ) = 0.037
21 Error(h1) = 0.818 x ( 1 - 0.818 ) x ( 1.5 x 0.037 ) = 0.008
22 Error(h2) = 0.953 x ( 1 - 0.953 ) x ( -1.0 x 0.037 ) = -0.002
23
24
25 Step 3: Learn
26
27 W(h0, y) = 1.0 + (0.5 * 0.037 * 1) = 1.018
28 W(h1, y) = 1.5 + (0.5 * 0.037 * 0.818) = 1.515
29 W(h2, y) = -1.0 + (0.5 * 0.037 * 0.953) = -0.982
30
31 W(I0, h1) = 1.0 + (0.5 * 0.008 * 1) = 1.004
32 W(I1, h1) = 1.0 + (0.5 * 0.008 * 0) = 1.0
33 W(I2, h1) = 0.5 + (0.5 * 0.008 * 1) = 0.504
34
35 W(I0, h2) = 1.0 + (0.5 * -0.002 * 1) = 0.999
36 W(I1, h2) = -1.0 + (0.5 * -0.002 * 0) = -1.0
37 W(I2, h2) = 2.0 + (0.5 * -0.002 * 1) = 1.999
38
39
40
41 Parameters at the end
42
43 [ 1.018  1.515 -0.982]
44 [[ 1.004  1.    0.504]
45  [ 0.999 -1.    1.999]]
46
47 Process finished with exit code 0
48

```

**Final network with the updated weights
(Manually drawn using the final weights)**



Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Analysis and Discussion:

We can see from the above visualization part that the output generated using the implemented program satisfies our objective of this project. In this project, we have understood from part A and part B regarding how Perceptron learning works by being able to implement a small model as well as the function of Backpropagation in Neural Network learning by implementing the steps executed for a single data instance.

Future Work:

In future, we can extend our program by implementing the capability to process Backpropagation in Neural Network learning by implementing the steps executed for whole dataset instead of a single data instance. Furthermore, we can extend our program to create a neural network capable of recognizing handwritten digits.

Source Code (Part A):

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```

1 from perceptron import Perceptron
2 from pandas import DataFrame
3
4
5 def run_perceptron():
6     print('\n\n***** AND *****\n\n')
7     p = Perceptron(2)
8     df = DataFrame([[0, 0], [0, 1], [1, 0], [1, 1]])
9     targets = [0, 0, 0, 1]
10    p.train(df, targets)
11    print('\n***** Classify *****')
12    p.classify([0, 1])
13
14    print('\n\n***** OR *****\n\n')
15    p = Perceptron(2)
16    df = DataFrame([[0, 0], [0, 1], [1, 0], [1, 1]])
17    targets = [0, 1, 1, 1]
18    p.train(df, targets)
19    print('\n***** Classify *****')
20    p.classify([0, 1])
21
22    print('\n\n***** NAND *****\n\n')
23    p = Perceptron(2)
24    df = DataFrame([[0, 0], [0, 1], [1, 0], [1, 1]])
25    targets = [1, 1, 1, 0]
26    p.train(df, targets)
27    print('\n***** Classify *****')
28    p.classify([0, 1])
29
30    print('\n\n***** NOR *****\n\n')
31    p = Perceptron(2)
32    df = DataFrame([[0, 0], [0, 1], [1, 0], [1, 1]])
33    targets = [1, 0, 0, 0]
34    p.train(df, targets)
35    print('\n***** Classify *****')
36    p.classify([0, 1])
37
38
39 if __name__ == '__main__':
40     run_perceptron()
41

```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```

1 import numpy as np
2 import pandas as pd
3 from random import seed, random
4
5
6 class Perceptron:
7     __weights = np.zeros((1, 3))
8     __epoch = 0
9     __learning_rate = 0.5
10
11     def __init__(self, inputs: int):
12         seed(1)
13         self.__weights = np.array([round(random(), 2) for i in range(0, inputs + 1)])
14         #self.__weights = np.array([0.01, 0.08, 0.08])
15         print(f'Initial Weights: {self.__weights[0]} {self.__weights[1:]}')
16
17     def __default_step_function(self, inputs):
18         product = self.__weights * inputs
19         return 1 if product.sum() > 0 else 0
20
21     def __default_update_parameters(self, inputs, t, y):
22         mult = self.__learning_rate * (t - y)
23         changes = np.array(inputs) * mult
24         self.__weights += changes
25
26     def set_hyper_parameters(self, learning_rate=0.5):
27         self.__learning_rate = learning_rate
28
29     def train(self, df: pd.DataFrame, targets, epochs=None):
30         if len(df) != len(targets):
31             raise Exception("Data and Targets don't have same number of rows.")
32
33         converged = False
34         while (epochs is None and not converged) or (epochs is not None and epochs != self.__epoch):
35
36             print('\n*****\n')
37             self.__epoch += 1
38             print(f'Epoch #{self.__epoch}')
39
40             correct_predictions = 0
41             for i in df.index:
42                 inputs = df.loc[i].to_list()
43                 print(f'\ninputs: {inputs}')
44                 print(f'weights: {self.__weights[0]} {self.__weights[1:]}')
45
46                 inputs.insert(0, 1)
47                 y = self.__default_step_function(inputs)
48                 print(f'y={y}\tt={targets[i]} ==> ', end='')
49
50                 if y == targets[i]:
51                     print('correct')
52                     correct_predictions += 1
53                 else:
54                     print('incorrect')
55                     # update weights
56                     self.__default_update_parameters(inputs, targets[i], y)
57
58             converged = correct_predictions == len(df)
59
60             print(f'\n\nSolution took {self.__epoch} epochs.')
61             print(f'Final weights: {self.__weights[0]} {self.__weights[1:]}')
62
63     def classify(self, inputs):
64         print(f'\ninputs: {inputs}')
65         print(f'weights: {self.__weights[0]} {self.__weights[1:]}')
66         ins = inputs.copy()
67         ins.insert(0, 1)
68         y = self.__default_step_function(ins)
69         print(f'Prediction: {y}')
70         return y
71

```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Source Code (Part B):

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```

1 import numpy as np
2 from pandas import DataFrame
3 import math
4
5 learning_rate = 0.5
6
7 first_layer_weights = np.array([
8     [1, 1, 0.5],
9     [1, -1, 2]
10 ])
11
12 second_layer_weights = np.array(
13     [1, 1.5, -1]
14 )
15
16
17 def feed_forward(inputs, target):
18     h, w = first_layer_weights.shape
19     hiddens = [0]*h
20
21     for l in range(0, h):
22         print(f'Sigma(h{l + 1}) = ' + '{', end='')
23         sign = ''
24         for i in range(0, len(inputs)):
25             print(f'{sign}({first_layer_weights[l][i]}*{inputs[i]}', end='')
26             sign = '+'
27         print(' } = ', end='')
28
29         sig_h1 = np.dot(first_layer_weights[l], inputs)
30         print(sig_h1, end='')
31
32         hiddens[l] = round((1 / (1 + math.e ** -sig_h1)), 3)
33         print(f'    h{l + 1} = {hiddens[l]}')
34
35     hiddens.insert(0, 1)
36     print('Sigma(y) = {', end='')
37     print(f'({second_layer_weights[0]}*{hiddens[0]}', end='')
38     print(f'+({second_layer_weights[1]}*{hiddens[1]}', end='')
39     print(f'+({second_layer_weights[2]}*{hiddens[2]}', end='')
40     print(' } = ', end='')
41
42     y = np.dot(second_layer_weights, hiddens)
43     hiddens[0] = (round((1/(1+math.e**-y)), 3))
44
45     print(f'{y}', end='')
46     print(f'    y = {hiddens[0]}')
47
48     E_net = round(0.5 * (target - hiddens[0]) ** 2, 3)
49     print(f'Total error in network E = (0.5x({target}-{hiddens[0]})^2) = {E_net}')
50
51     return hiddens, E_net
52
53
54 def backpropagate_errors(hiddens, target):
55     errors = [0]*len(hiddens)
56     errors[0] = round(hiddens[0]*(1-hiddens[0])*(target-hiddens[0]), 3)
57     print(f'Error(y) = {hiddens[0]} x ( 1 - {hiddens[0]} ) x ( {target} - {hiddens[0]} ) = {errors[0]}')
58
59     for i in range(1, len(hiddens)):
60         errors[i] = round(hiddens[i]*(1-hiddens[i])*(second_layer_weights[i]*errors[0]), 3)
61         print(f'Error(h{i}) = {hiddens[i]} x ( 1 - {hiddens[i]} ) x ( {second_layer_weights[i]} x {errors[0]} ) = {
errors[i]}')
62
63     return errors
64
65
66 def learn(hiddens, errors, inputs):
67     hiddens.insert(0, 1)
68
69     for i in range(0, len(second_layer_weights)):
70         nw = round(second_layer_weights[i] + (learning_rate * errors[0] * hiddens[i]), 3)
71         print(f'W(h{i}, y) = {second_layer_weights[i]} + ({learning_rate} * {errors[0]} * {hiddens[i]}) = {nw}')
72         second_layer_weights[i] = nw
73
74     print('')
75
76     h, w = first_layer_weights.shape
77     for i in range(0, h):
78         for j in range(0, w):
79             nw = first_layer_weights[i][j] + (learning_rate * errors[i+1] * inputs[j])

```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```
80         print(f'W(I{j}, h{i+1}) = {first_layer_weights[i][j]} + ({learning_rate} * {errors[i+1]} * {inputs[j]}) = {
nw}')
81         first_layer_weights[i][j] = nw
82         print('')
83
84
85 target = 1
86 inputs = [1, 0, 1]
87 print('Step 1: Feed the Inputs forward\n')
88 val_at_nodes, E_net = feed_forward(inputs, target)
89
90 print('\n\nStep 2: Backpropagate the errors\n')
91 errors = backpropagate_errors(val_at_nodes, target)
92
93 print('\n\nStep 3: Learn\n')
94 learn(val_at_nodes[1:], errors, inputs)
95
96 print('\n\nParameters at the end\n')
97 print(second_layer_weights)
98 print(first_layer_weights)
99
100
101
```

Thank You!!!