

CMPE 260 RL Midterm Exam

Instructor: Jahan Ghofraniha

Student Name: Anusha Gangasani

ID: 015513160

Problem 1 (60 pts):

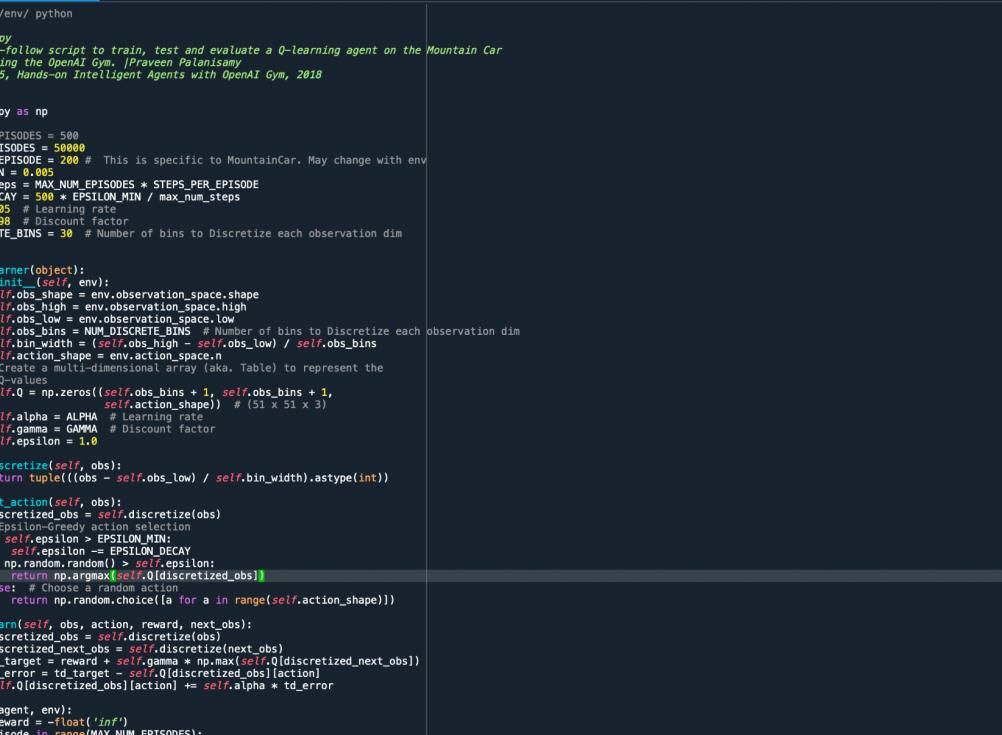
- ## 1. Q-Learning for mountain car

Reference:

https://sjsu.instructure.com/courses/1529630/files/69364522?module_item_id=1

3628553

Code:



The screenshot shows the Spyder Python IDE interface with the following details:

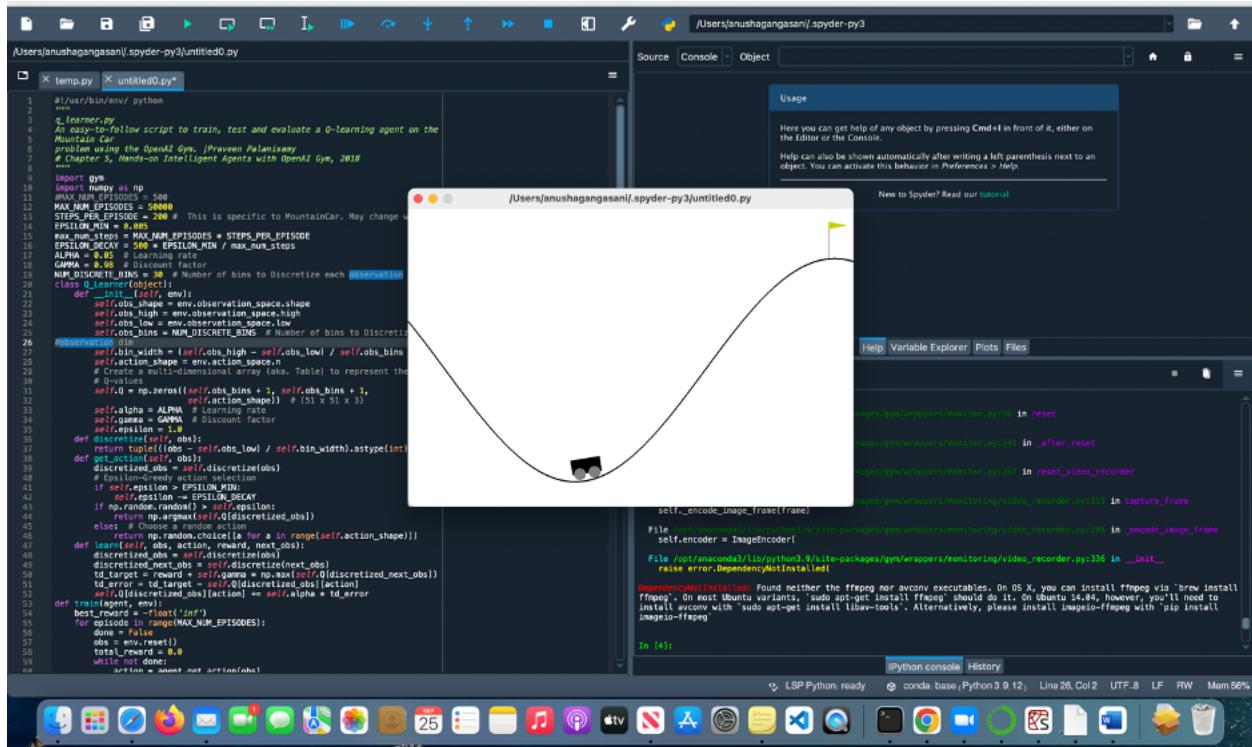
- Title Bar:** /Users/anushagangasani/.spyder-py3/untitled0.py
- Toolbar:** Standard file operations (New, Open, Save, Print, Find, Copy, Paste, etc.) and a Run button.
- Code Editor:** The main window displays the Python code for a Q-Learning agent. The code includes imports for gym and numpy, defines constants for environment parameters, and implements the QLearner class with methods for initialization, action selection, discretization, learning, and training.
- Status Bar:** Shows the current file path as /Users/anushagangasani/.spyder-py3/untitled0.py.

```
1 #!/usr/bin/env python
2 """
3     q_learner.py
4     An easy-to-follow script to train, test and evaluate a Q-learning agent on the Mountain Car
5     problem using the OpenAI Gym. |Praveen Palansamy
6     # Chapter 5, Hands-on Intelligent Agents with OpenAI Gym, 2018
7 """
8 import gym
9 import numpy as np
10
11 #MAX_NUM_EPISODES = 500
12 MAX_NUM_EPISODES = 5000
13 STEPS_PER_EPISODE = 200 # This is specific to MountainCar. May change with env
14 EPSILON_MIN = 0.005
15 max_num_steps = MAX_NUM_EPISODES * STEPS_PER_EPISODE
16 EPSILON_DECAY = 500 * EPSILON_MIN / max_num_steps
17 ALPHA = 0.05 # Learning rate
18 GAMMA = 0.98 # Discount factor
19 NUM_DISCRETE_BINS = 30 # Number of bins to Discretize each observation dim
20
21
22 class QLearner(object):
23     def __init__(self, env):
24         self.obs_shape = env.observation_space.shape
25         self.obs_high = env.observation_space.high
26         self.obs_low = env.observation_space.low
27         self.obs_bins = NUM_DISCRETE_BINS # Number of bins to Discretize each observation dim
28         self.bin_width = (self.obs_high - self.obs_low) / self.obs_bins
29         self.action_shape = env.action_space.n
30         # Create a multi-dimensional array (aka. Table) to represent the
31         # Q-values
32         self.Q = np.zeros((self.obs_bins + 1, self.obs_bins + 1,
33                           self.action_shape)) # (51 x 51 x 3)
34         self.alpha = ALPHA # Learning rate
35         self.gamma = GAMMA # Discount factor
36         self.epsilon = 1.0
37
38     def discretize(self, obs):
39         return tuple(((obs - self.obs_low) / self.bin_width).astype(int))
40
41     def get_action(self, obs):
42         discretized_obs = self.discretize(obs)
43         # Epsilon-Greedy action selection
44         if self.epsilon > EPSILON_MIN:
45             self.epsilon -= EPSILON_DECAY
46         if np.random.random() > self.epsilon:
47             return np.argmax(self.Q[discretized_obs])
48         else: # Choose a random action
49             return np.random.choice([a for a in range(self.action_shape)])
50
51     def learn(self, obs, action, reward, next_obs):
52         discretized_obs = self.discretize(obs)
53         discretized_next_obs = self.discretize(next_obs)
54         td_target = reward + self.gamma * np.max(self.Q[discretized_next_obs])
55         td_error = td_target - self.Q[discretized_obs][action]
56         self.Q[discretized_obs][action] += self.alpha * td_error
57
58     def train(self, agent, env):
59         best_reward = float('-inf')
60         for episode in range(MAX_NUM_EPISODES):
61             done = False
62             obs = env.reset()
63             total_reward = 0.0
64             while not done:
65                 action = agent.get_action(obs)
66                 next_obs, reward, done, info = env.step(action)
67                 agent.learn(obs, action, reward, next_obs)
68                 obs = next_obs
69                 total_reward += reward
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
140
141
142
143
144
145
146
147
148
149
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
217
218
219
219
220
221
222
223
224
225
226
227
227
228
229
229
230
231
232
233
234
235
236
237
237
238
239
239
240
241
241
242
243
244
245
246
247
247
248
249
249
250
251
252
253
254
255
256
256
257
258
258
259
259
260
261
262
263
264
265
266
267
267
268
269
269
270
271
272
273
274
275
276
277
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
307
308
309
309
310
311
312
313
314
315
316
316
317
318
318
319
319
320
321
322
323
324
325
326
326
327
328
328
329
329
330
331
332
333
334
335
336
336
337
338
338
339
339
340
341
342
343
344
345
345
346
347
347
348
348
349
349
350
351
352
353
354
355
355
356
357
357
358
358
359
359
360
361
362
363
364
365
365
366
367
367
368
368
369
369
370
371
372
373
374
375
375
376
377
377
378
378
379
379
380
381
382
383
384
385
385
386
387
387
388
388
389
389
390
391
392
393
394
395
395
396
397
397
398
398
399
399
400
401
402
403
404
404
405
406
406
407
407
408
408
409
409
410
411
412
413
414
414
415
415
416
416
417
417
418
418
419
419
420
421
422
423
424
425
425
426
426
427
427
428
428
429
429
430
431
432
433
434
434
435
435
436
436
437
437
438
438
439
439
440
441
442
443
444
444
445
445
446
446
447
447
448
448
449
449
450
451
452
453
454
454
455
455
456
456
457
457
458
458
459
459
460
461
462
463
464
464
465
465
466
466
467
467
468
468
469
469
470
471
472
473
474
474
475
475
476
476
477
477
478
478
479
479
480
481
482
483
484
484
485
485
486
486
487
487
488
488
489
489
490
491
492
493
494
494
495
495
496
496
497
497
498
498
499
499
500
501
502
503
504
504
505
505
506
506
507
507
508
508
509
509
510
511
512
513
514
514
515
515
516
516
517
517
518
518
519
519
520
521
522
523
524
524
525
525
526
526
527
527
528
528
529
529
530
531
532
533
534
534
535
535
536
536
537
537
538
538
539
539
540
541
542
543
544
544
545
545
546
546
547
547
548
548
549
549
550
551
552
553
554
554
555
555
556
556
557
557
558
558
559
559
560
561
562
563
564
564
565
565
566
566
567
567
568
568
569
569
570
571
572
573
574
574
575
575
576
576
577
577
578
578
579
579
580
581
582
583
584
584
585
585
586
586
587
587
588
588
589
589
590
591
592
593
594
594
595
595
596
596
597
597
598
598
599
599
600
601
602
603
604
604
605
605
606
606
607
607
608
608
609
609
610
611
612
613
614
614
615
615
616
616
617
617
618
618
619
619
620
621
622
623
624
624
625
625
626
626
627
627
628
628
629
629
630
631
632
633
634
634
635
635
636
636
637
637
638
638
639
639
640
641
642
643
644
644
645
645
646
646
647
647
648
648
649
649
650
651
652
653
654
654
655
655
656
656
657
657
658
658
659
659
660
661
662
663
664
664
665
665
666
666
667
667
668
668
669
669
670
671
672
673
674
674
675
675
676
676
677
677
678
678
679
679
680
681
682
683
684
684
685
685
686
686
687
687
688
688
689
689
690
691
692
693
694
694
695
695
696
696
697
697
698
698
699
699
700
701
702
703
704
704
705
705
706
706
707
707
708
708
709
709
710
711
712
713
714
714
715
715
716
716
717
717
718
718
719
719
720
721
722
723
724
724
725
725
726
726
727
727
728
728
729
729
730
731
732
733
734
734
735
735
736
736
737
737
738
738
739
739
740
741
742
743
744
744
745
745
746
746
747
747
748
748
749
749
750
751
752
753
754
754
755
755
756
756
757
757
758
758
759
759
760
761
762
763
764
764
765
765
766
766
767
767
768
768
769
769
770
771
772
773
774
774
775
775
776
776
777
777
778
778
779
779
780
781
782
783
784
784
785
785
786
786
787
787
788
788
789
789
790
791
792
793
794
794
795
795
796
796
797
797
798
798
799
799
800
801
802
803
804
804
805
805
806
806
807
807
808
808
809
809
810
811
812
813
814
814
815
815
816
816
817
817
818
818
819
819
820
821
822
823
824
824
825
825
826
826
827
827
828
828
829
829
830
831
832
833
834
834
835
835
836
836
837
837
838
838
839
839
840
841
842
843
844
844
845
845
846
846
847
847
848
848
849
849
850
851
852
853
854
854
855
855
856
856
857
857
858
858
859
859
860
861
862
863
864
864
865
865
866
866
867
867
868
868
869
869
870
871
872
873
874
874
875
875
876
876
877
877
878
878
879
879
880
881
882
883
884
884
885
885
886
886
887
887
888
888
889
889
890
891
892
893
894
894
895
895
896
896
897
897
898
898
899
899
900
901
902
903
904
904
905
905
906
906
907
907
908
908
909
909
910
911
912
913
914
914
915
915
916
916
917
917
918
918
919
919
920
921
922
923
924
924
925
925
926
926
927
927
928
928
929
929
930
931
932
933
934
934
935
935
936
936
937
937
938
938
939
939
940
941
942
943
944
944
945
945
946
946
947
947
948
948
949
949
950
951
952
953
954
954
955
955
956
956
957
957
958
958
959
959
960
961
962
963
964
964
965
965
966
966
967
967
968
968
969
969
970
971
972
973
974
974
975
975
976
976
977
977
978
978
979
979
980
981
982
983
984
984
985
985
986
986
987
987
988
988
989
989
990
991
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1001
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1011
1012
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1021
1022
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1031
1032
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1041
1042
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1051
1052
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1061
1062
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1071
1072
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1081
1082
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1091
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1101
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1111
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1121
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1131
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1141
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1151
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1161
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1171
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1181
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
```

The screenshot shows the Spyder IDE interface with a Python script named `temp.py` open. The code implements a Q-Learning agent for the MountainCar-v0 environment. It includes methods for discretizing observations, selecting actions based on epsilon-greedy policy, learning from experiences, training the agent, and testing it. The code also handles Gym environments and uses GymMonitor to record video during training.

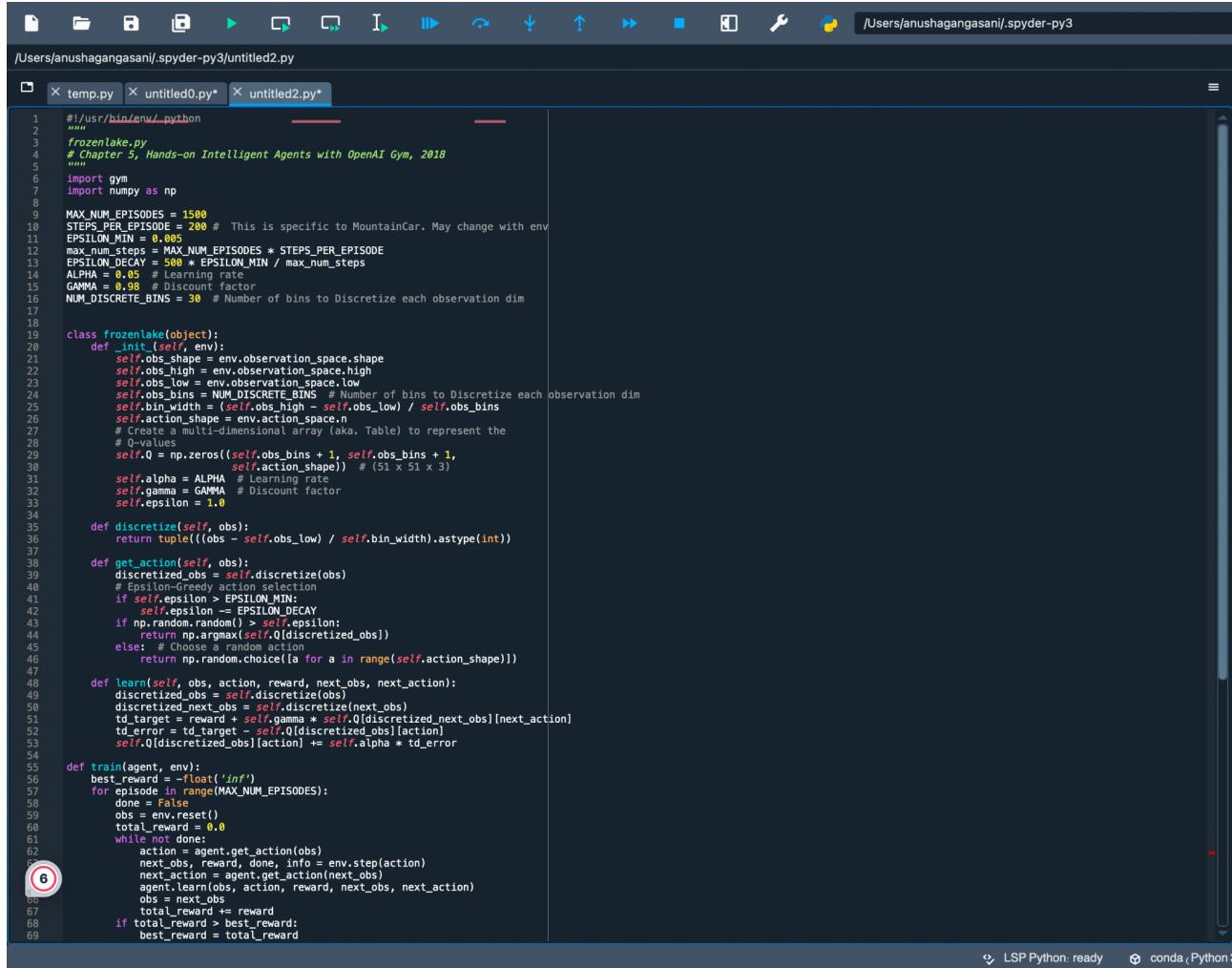
```
32     self.Q = np.zeros((self.obs_bins + 1, self.obs_bins + 1,
33                        self.action_shape)) # (51 x 51 x 3)
34     self.alpha = ALPHA # Learning rate
35     self.gamma = GAMMA # Discount factor
36     self.epsilon = 1.0
37
38     def discretize(self, obs):
39         return tuple((obs - self.obs_low) / self.bin_width).astype(int)
40
41     def get_action(self, obs):
42         discretized_obs = self.discretize(obs)
43         # Epsilon-Greedy action selection
44         if self.epsilon > EPSILON_MIN:
45             self.epsilon -= EPSILON_DECAY
46         if np.random.random() > self.epsilon:
47             return np.argmax(self.Q[discretized_obs])
48         else: # Choose a random action
49             return np.random.choice([a for a in range(self.action_shape)])
50
51     def learn(self, obs, action, reward, next_obs):
52         discretized_obs = self.discretize(obs)
53         discretized_next_obs = self.discretize(next_obs)
54         td_target = reward + self.gamma * np.max(self.Q[discretized_next_obs])
55         td_error = td_target - self.Q[discretized_obs][action]
56         self.Q[discretized_obs][action] += self.alpha * td_error
57
58     def train(agent, env):
59         best_reward = -float('inf')
60         for episode in range(MAX_NUM_EPISODES):
61             done = False
62             obs = env.reset()
63             total_reward = 0.0
64             while not done:
65                 action = agent.get_action(obs)
66                 next_obs, reward, done, info = env.step(action)
67                 agent.learn(obs, action, reward, next_obs)
68                 obs = next_obs
69                 total_reward += reward
70             if total_reward > best_reward:
71                 best_reward = total_reward
72             print("Episode:{} reward:{} eps:{} total_reward:{} best_reward:{} agent.epsilon:{}".format(episode, reward, episode, total_reward, best_reward, agent.epsilon))
73
74     # Return the trained policy
75     return np.argmax(agent.Q, axis=2)
76
77
78     def test(agent, env, policy):
79         done = False
80         obs = env.reset()
81         total_reward = 0.0
82         while not done:
83             action = policy[agent.discretize(obs)]
84             next_obs, reward, done, info = env.step(action)
85             obs = next_obs
86             total_reward += reward
87         return total_reward
88
89
90     if __name__ == "__main__":
91         env = gym.make("MountainCar-v0")
92         agent = Q_Learner(env)
93         learned_policy = train(agent, env)
94         # Use GymMonitor wrapper to evaluate the agent and record video
95         gym_monitor_path = "./gym_monitor_output"
96         env = gym.wrappers.Monitor(env, gym_monitor_path, force=True)
97         for _ in range(1000):
98             test(agent, env, learned_policy)
99         env.close()
```

Output:



2. Q-iteration sample code based on frozen lake environment

Code:



The screenshot shows the Spyder IDE interface with the following details:

- Title Bar:** /Users/anushagangasani/.spyder-py3/untitled2.py
- File List:** Shows three files: temp.py, untitled0.py*, and untitled2.py*.
- Code Editor:** Displays the content of the untitled2.py* file, which is a Python script for a reinforcement learning task. The code includes imports for gym and numpy, defines constants for episodes, steps per episode, epsilon decay, alpha, gamma, and the number of discrete bins, and implements a frozenlake class with methods for discretizing observations, selecting actions, learning, and training an agent.
- Annotations:** A red circle highlights the number "6" at the bottom left of the code editor area.
- Status Bar:** Shows "LSP Python: ready" and "conda_Python".

```
#!/usr/bin/env python
"""
frozenlake.py
# Chapter 5, Hands-on Intelligent Agents with OpenAI Gym, 2018
"""

import gym
import numpy as np

MAX_NUM_EPISODES = 1500
STEPS_PER_EPISODE = 200 # This is specific to MountainCar. May change with env
EPSILON_MIN = 0.005
max_num_steps = MAX_NUM_EPISODES * STEPS_PER_EPISODE
EPSILON_DECAY = 500 * EPSILON_MIN / max_num_steps
ALPHA = 0.05 # Learning rate
GAMMA = 0.98 # Discount factor
NUM_DISCRETE_BINS = 30 # Number of bins to Discretize each observation dim

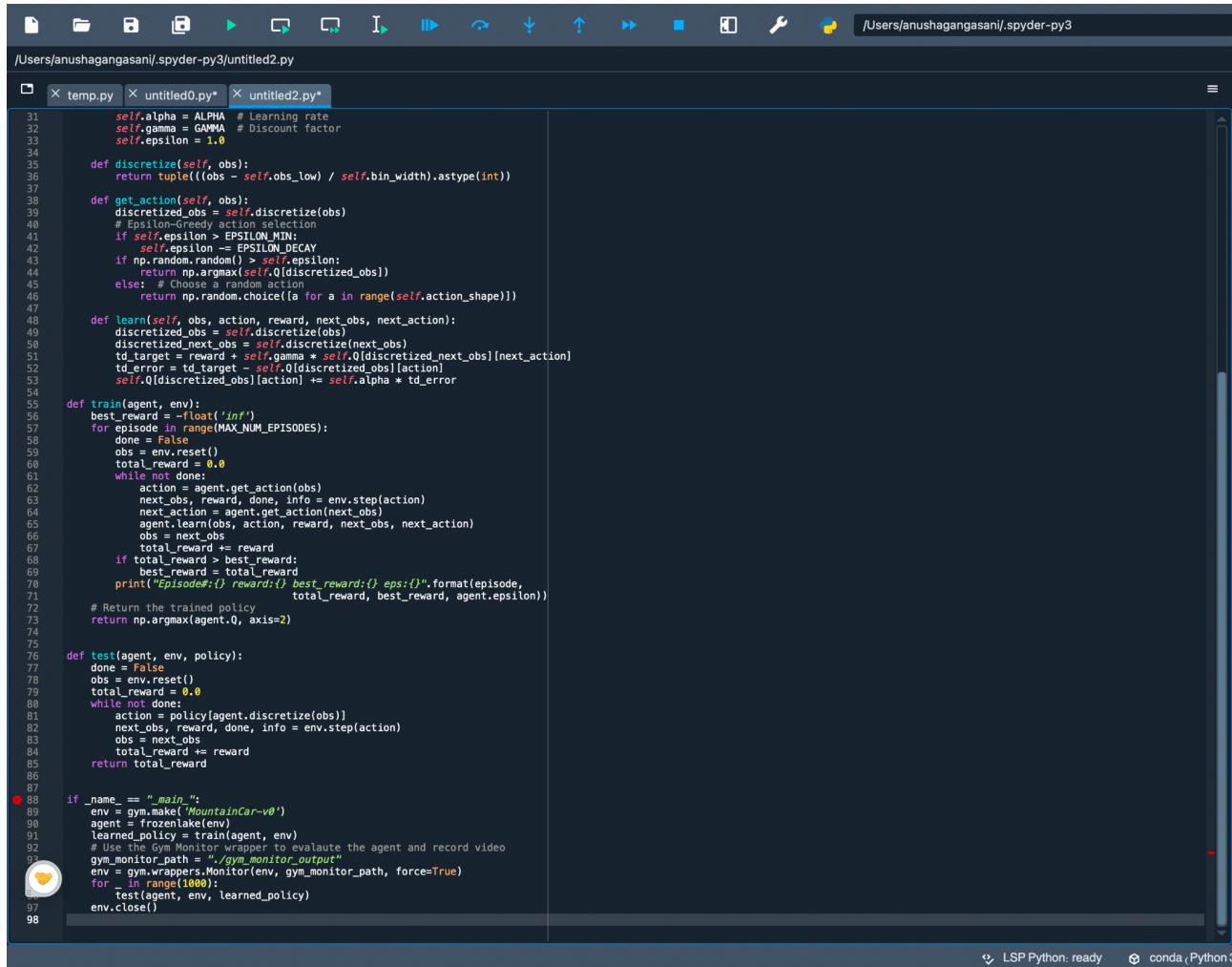
class frozenlake(object):
    def __init__(self, env):
        self.obs_shape = env.observation_space.shape
        self.obs_high = env.observation_space.high
        self.obs_low = env.observation_space.low
        self.num_discrete_bins = NUM_DISCRETE_BINS # Number of bins to Discretize each observation dim
        self.bin_width = (self.obs_high - self.obs_low) / self.obs_bins
        self.action_shape = env.action_space.n
        # Create a multi-dimensional array (aka. Table) to represent the
        # 0-values
        self.Q = np.zeros((self.obs_bins + 1, self.action_shape)) # (51 x 51 x 3)
        self.alpha = ALPHA # Learning rate
        self.gamma = GAMMA # Discount factor
        self.epsilon = 1.0

    def discretize(self, obs):
        return tuple(((obs - self.obs_low) / self.bin_width).astype(int))

    def get_action(self, obs):
        discretized_obs = self.discretize(obs)
        # Epsilon-Greedy action selection
        if self.epsilon > EPSILON_MIN:
            self.epsilon -= EPSILON_DECAY
        if np.random.rand() < self.epsilon:
            return np.argmax(self.Q[discretized_obs])
        else: # Choose a random action
            return np.random.choice([a for a in range(self.action_shape)])

    def learn(self, obs, action, reward, next_obs, next_action):
        discretized_obs = self.discretize(obs)
        discretized_next_obs = self.discretize(next_obs)
        td_target = reward + self.gamma * self.Q[discretized_next_obs][next_action]
        td_error = td_target - self.Q[discretized_obs][action]
        self.Q[discretized_obs][action] += self.alpha * td_error

    def train(self, env):
        best_reward = -float('inf')
        for episode in range(MAX_NUM_EPISODES):
            done = False
            obs = env.reset()
            total_reward = 0.0
            while not done:
                action = self.get_action(obs)
                next_obs, reward, done, info = env.step(action)
                self.learn(obs, action, reward, next_obs, next_action)
                obs = next_obs
                total_reward += reward
            if total_reward > best_reward:
                best_reward = total_reward
```



The screenshot shows the Spyder Python IDE interface. The top bar displays the path: /Users/anushagangasani/.spyder-py3/untitled2.py. The main area contains a code editor with the following Python script:

```
31     self.alpha = ALPHA # Learning rate
32     self.gamma = GAMMA # Discount factor
33     self.epsilon = 1.0
34
35     def discretize(self, obs):
36         return tuple((obs - self.obs_low) / self.bin_width).astype(int)
37
38     def get_action(self, obs):
39         discretized_obs = self.discretize(obs)
40         # Epsilon-Greedy action selection
41         if self.epsilon > EPSILON_MIN:
42             td_error = np.random.normal(0, self.error_decay)
43             if np.random.random() > self.epsilon:
44                 return np.argmax(self.Q[discretized_obs])
45             else: # Choose a random action
46                 return np.random.choice([a for a in range(self.action_shape)])
47
48     def learn(self, obs, action, reward, next_obs, next_action):
49         discretized_obs = self.discretize(obs)
50         discretized_next_obs = self.discretize(next_obs)
51         td_target = reward + self.gamma * self.Q[discretized_next_obs][next_action]
52         td_error = td_target - self.Q[discretized_obs][action]
53         self.Q[discretized_obs][action] += self.alpha * td_error
54
55     def train(agent, env):
56         best_reward = -float('inf')
57         for episode in range(MAX_NUM_EPISODES):
58             done = False
59             obs = env.reset()
60             total_reward = 0.0
61             while not done:
62                 action = agent.get_action(obs)
63                 next_obs, reward, done, info = env.step(action)
64                 next_action = agent.get_action(next_obs)
65                 agent.learn(obs, action, reward, next_obs, next_action)
66                 obs = next_obs
67                 total_reward += reward
68             if total_reward > best_reward:
69                 best_reward = total_reward
70             print("Episode:{} reward:{} best_reward:{} eps:{}".
71                  format(episode, total_reward, best_reward, agent.epsilon))
72
73         # Return the trained policy
74         return np.argmax(agent.Q, axis=2)
75
76     def test(agent, env, policy):
77         done = False
78         obs = env.reset()
79         total_reward = 0.0
80         while not done:
81             action = policy[agent.discretize(obs)]
82             next_obs, reward, done, info = env.step(action)
83             obs = next_obs
84             total_reward += reward
85         return total_reward
86
87
88 if __name__ == "__main__":
89     env = gym.make('MountainCar-v0')
90     agent = FrozenLakeEnv()
91     learned_policy = train(agent, env)
92     # Use the Gym Monitor wrapper to evaluate the agent and record video
93     gym_monitor_path = "/gym_monitor_output"
94     env = gym.wrappers.Monitor(env, gym_monitor_path, force=True)
95     for i in range(1000):
96         test(agent, env, learned_policy)
97     env.close()
```

Output:

The screenshot shows the Spyder Python IDE interface. On the left, there's a code editor with several tabs: 'temp.py', 'untitled0.py*', and 'sarsa.py'. The 'sarsa.py' tab contains Python code for a SARSA algorithm. In the center, a 3D plot visualizes a curved surface representing a value function or policy. On the right, a 'Usage' help panel is open, and at the bottom, the Mac OS X dock is visible.

```

15 max_num_steps = MAX_NUM_EPISODES * STEPS_PER_EPISODE
16 max_steps = 51 * 51 * 3 * EPISODE_N / max_num_steps
17 ALPHA = 0.05 # Learning rate
18 GAMMA = 0.98 # Discount factor
19 NUM_DISCRETE_BINS = 30 # Number of bins to Discretize each observation dim
20
21
22 class SARSA(object):
23     def __init__(self, env):
24         self.obs_shape = env.observation_space.shape
25         self.obs_high = env.observation_space.high
26         self.obs_low = env.observation_space.low
27         self.obs_bins = NUM_DISCRETE_BINS # Number of bins to Discretize each observation dim
28         self.bin_width = (self.obs_high - self.obs_low) / self.obs_bins
29         self.action_shape = env.action_space.n
30         # Create a multi-dimensional array (aka. Table) to represent the
31         # Q-values
32         self.Q = np.zeros((self.obs_bins + 1, self.obs_bins + 1,
33                           self.action_shape)) # (51 x 51 x 3)
34         self.alpha = ALPHA # Learning rate
35         self.gamma = GAMMA # Discount factor
36         self.epsilon = 1.0
37
38     def discretize(self, obs):
39         return tuple(((obs - self.obs_low) / self.bin_width).astype(int))
40
41     def get_action(self, obs):
42         discretized_obs = self.discretize(obs)
43         if self.epsilon > EPSILON_MIN:
44             self.epsilon -= EPSILON_DECAY
45         if np.random.rand() < self.epsilon:
46             return np.argmax(self.Q[discretized_obs])
47         else: # Choose a random action
48             return np.random.choice([a for a in range(self.action_shape)])
49
50     def learn(self, obs, action, reward, next_obs, next_action):
51         discretized_obs = self.discretize(obs)
52         discretized_next_obs = self.discretize(next_obs)
53         td_target = reward + self.gamma * self.Q[discretized_next_obs][next_action]
54         td_error = td_target - self.Q[discretized_obs][action]
55         self.Q[discretized_obs][action] += self.alpha * td_error
56
57     def train(self, env):
58         best_reward = -float('inf')
59         for episode in range(MAX_NUM_EPISODES):
60             done = False
61             obs = env.reset()
62             total_reward = 0.0
63             while not done:
64                 action = agent.get_action(obs)
65                 next_obs, reward, done, info = env.step(action)
66                 next_obs, reward, done, info = env.step(action)
67                 agent.learn(obs, action, reward, next_obs, next_action)
68                 obs = next_obs
69                 total_reward += reward
70             if total_reward > best_reward:
71                 best_reward = total_reward
72
73
74
75
76
77
78
79
79
80
80
81
81
82
82
83
83
84
84
85
85
86
86
87
87
88
88
89
89
90
90
91
91
92
92
93
93
94
94
95
95
96
96
97
97
98
98
99
99
100
100
101
101
102
102
103
103
104
104
105
105
106
106
107
107
108
108
109
109
110
110
111
111
112
112
113
113
114
114
115
115
116
116
117
117
118
118
119
119
120
120
121
121
122
122
123
123
124
124
125
125
126
126
127
127
128
128
129
129
130
130
131
131
132
132
133
133
134
134
135
135
136
136
137
137
138
138
139
139
140
140
141
141
142
142
143
143
144
144
145
145
146
146
147
147
148
148
149
149
150
150
151
151
152
152
153
153
154
154
155
155
156
156
157
157
158
158
159
159
160
160
161
161
162
162
163
163
164
164
165
165
166
166
167
167
168
168
169
169
170
170
171
171
172
172
173
173
174
174
175
175
176
176
177
177
178
178
179
179
180
180
181
181
182
182
183
183
184
184
185
185
186
186
187
187
188
188
189
189
190
190
191
191
192
192
193
193
194
194
195
195
196
196
197
197
198
198
199
199
200
200
201
201
202
202
203
203
204
204
205
205
206
206
207
207
208
208
209
209
210
210
211
211
212
212
213
213
214
214
215
215
216
216
217
217
218
218
219
219
220
220
221
221
222
222
223
223
224
224
225
225
226
226
227
227
228
228
229
229
230
230
231
231
232
232
233
233
234
234
235
235
236
236
237
237
238
238
239
239
240
240
241
241
242
242
243
243
244
244
245
245
246
246
247
247
248
248
249
249
250
250
251
251
252
252
253
253
254
254
255
255
256
256
257
257
258
258
259
259
260
260
261
261
262
262
263
263
264
264
265
265
266
266
267
267
268
268
269
269
270
270
271
271
272
272
273
273
274
274
275
275
276
276
277
277
278
278
279
279
280
280
281
281
282
282
283
283
284
284
285
285
286
286
287
287
288
288
289
289
290
290
291
291
292
292
293
293
294
294
295
295
296
296
297
297
298
298
299
299
300
300
301
301
302
302
303
303
304
304
305
305
306
306
307
307
308
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
131
```

4. https://github.com/gangasani-anusha/RL_Midterm.git

Problem 2 (20 pts):

Reference:

<https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/>

https://sjsu.instructure.com/courses/1529630/files/69364473?module_item_id=13628585

How does Batch Normalization work?

Batch Normalization is a process to make neural networks faster and more stable by adding extra layers in a DNN. It works in a two-step process. Initially, the input is normalized, and then offsetting, and rescaling is executed.

1. Normalization of the Input:

Normalization is a process of converting the data to get the mean as zero and the standard deviation as one. To calculate the mean of hidden activation:

$$\mu = \frac{1}{m} \sum h_i$$

Here m is the number of neurons at layer h.

Once the mean is obtained, let's calculate the standard deviation of the hidden activations.

$$\sigma = \left[\frac{1}{m} \sum (h_i - \mu)^2 \right]^{1/2}$$

We will normalize the hidden activations with the derived mean and standard deviation. The smoothing term(ϵ) satisfies the numerical stability within the operation. We normalize as follows:

$$h_{i(\text{norm})} = \frac{(h_i - \mu)}{\sigma + \epsilon}$$

2. Rescaling of offsetting:

The final step is to rescale and offset the input. Here we use two important components gamma and beta of the batch normalization algorithm. These parameters are used for shifting(beta) and re-scaling(gamma) of the vector from the previous operations.

$$h_i = \gamma h_{i(\text{norm})} + \beta$$

These two parameters can be learned, and a neural network ensures that the best values are employed during training. This will make it possible to accurately normalize each batch.

What kind of problem does Batch Normalization solve?

- By normalizing the hidden layer activation, batch normalization speeds up the training process.
- BN solves the problem of internal covariate shift. By this, the input is distributed for every layer of mean and standard deviation. It helps by making the data flow between the layers of the neural network look, this means we can use a higher learning rate. It also has a regularizing effect which means we can remove dropout.
- BN smoothens the loss function which optimizes the model parameters to improve the training speed of the model.

Explain how BN works for the test data?

In Renormalization, the best practice of BN work for test data is to normalize the activations using the moving averages like mean and variance instead of mean and variance of minibatch. This depends only on a single input example instead of using a whole mini-batch.

$$y_{\text{inference}} = \frac{x - \mu}{\sigma} \cdot \gamma + \beta$$

Problem 3 (20 pts):

References:

https://sjsu.instructure.com/courses/1529630/files/69364531?module_item_id=13628595

<https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>

<https://uu.diva-portal.org/smash/get/diva2:1162949/FULLTEXT02.pdf>

We typically have three layers (Input, Hidden, and Output) depending on the image that is used for training convolutional neural networks. Here, we can actually change the processing's hidden layers to get rid of erroneous positive results. In this instance, executing these actions to enhance hidden layers after obtaining an image and applying CNN on it may eliminate or reduce the false positives.

To categorize it further in few steps:

- Altering the hidden layers in a rhombus format that is adjusting it in the descending order, where as for example it can be in the order of 726, 512, 256, 128 for the 1st to 4th hidden layer respectively.
- Increase the Epoch by 100 as more the iterations over training the more the efficiency could be increased.
- Increasing the dropout layer by 0.3 or training it by altering the percentage ranging from 0.2 to 0.9 that is 20% to 90%.
- If it comes to a complete dataset, it may be caused due to imbalanced data as well.
- In this case we can actually increase the F-score.
- If the images are TEM graded images that is low-resolution.
- Over-Sampling: Just like down-sampling but here we combine two or more classes to higher value average and training with the classifier again.
- Also lowering the NCC threshold may increase the number of candidates to analyze and automate the CNN again.
- In this case we can also down-sample the data: combining two different classes/image_features and reducing it to the minimum class and training with the classifier again.

As per these techniques/assumptions, there is a possibility to reduce the false positivity.

Remedies:

1) Cross Validation: train and test your model k-times on different subsets of training data and build up an estimate of the performance of a machine learning model on unseen data.

2 Regularization:

a) Max-Norm Regularization: for each neuron, it constrains the weights w of the incoming connections such that $\|w\|_2 \leq r$, where r is the max - norm hyperparameter and $\|\cdot\|_2$ is the ℓ_2 norm. Reducing r increases the amount of regularization and helps reduce Overfitting. This can be

implemented in keras as :

b) L2 regularization:

you can use ℓ_2 regularization to constrain a neural network's connection weights, and/or ℓ_1 regularization if you want a sparse model (with many weights equal to 0). Here is how to apply ℓ_2 regularization to a Keras layer's connection weights, using a regularization factor of 0.01.

The `l2()` function returns a regularizer that will be called at each step during training to compute the regularization loss. This is then added to the final loss. As you might expect, you can just use `keras.regularizers.l1()` if you want ℓ_1 regularization; if you want both ℓ_1 and ℓ_2 regularization, use `keras.regularizers.l1_l2()` (specifying both regularization factors)