

train.py | oct-11-train.py

1 Addition, 21 Deletions, 9 Changes

A train.py

```

1  import shutil
2  import os
3  import time
4  from datetime import datetime
5  import random
6  import argparse
7  import numpy as np
8  - import torchviz
9  from tqdm import tqdm
10
11  import torch
12  import torch.nn as nn
13  import torch.optim as optim
14  import torch.nn.functional as F
15  - from sklearn.metrics import roc_curve
16  from tensorboardX import SummaryWriter
17
18  from dataloader import MRDataset
19  from models.mrnet import MRNet
20
21  from sklearn import metrics
22  import csv
23  import utils as ut
24
25  - # torch.autograd.set_detect_anomaly(True)
26
27  def train_model(model, train_loader, epoch,
28  num_epochs, optimizer, writer, current_lr,
29  device, log_every=100):
30  """
31  Procedure to train a model on the
32  training set
33  """
34  model.train()
35
36  model = model.to(device)
37
38  y_preds = []
39  y_trues = []
40  losses = []
41
42  for i, (image, label, weight) in
43  enumerate(train_loader):
44
45  image = image.to(device)
46  label = label.to(device)
47  weight = weight.to(device)
48
49  prediction = model(image.float())
50
51  loss =
52  F.binary_cross_entropy_with_logits(prediction
53  , label, weight=weight)
54
55  optimizer.zero_grad()
56  loss.backward()
57  optimizer.step()
58
59  loss_value = loss.item()
60  losses.append(loss_value)
61

```

B oct-11-train.py

```

1  import shutil
2  import os
3  import time
4  from datetime import datetime
5  import random
6  import argparse
7  import numpy as np
8
9  from tqdm import tqdm
10
11  import torch
12  import torch.nn as nn
13  import torch.optim as optim
14  import torch.nn.functional as F
15
16  from tensorboardX import SummaryWriter
17
18  from dataloader import MRDataset
19  from models.mrnet import MRNet
20
21  from sklearn import metrics
22  import csv
23  import utils as ut
24
25
26  def train_model(model, train_loader, epoch,
27  num_epochs, optimizer, writer, current_lr,
28  device, log_every=100):
29  """
30  Procedure to train a model on the
31  training set
32  """
33  model.train()
34
35  model = model.to(device)
36
37  y_preds = []
38  y_trues = []
39  losses = []
40
41  for i, (image, label, weight) in
42  enumerate(train_loader):
43
44  image = image.to(device)
45  label = label.to(device)
46  weight = weight.to(device)
47
48  prediction = model(image.float())
49
50  loss =
51  F.binary_cross_entropy_with_logits(prediction
52  , label, weight=weight)
53
54
55  optimizer.zero_grad()
56  loss.backward()
57  optimizer.step()
58
59  loss_value = loss.item()
60  losses.append(loss_value)
61

```

A train.py

```

56         probas = torch.sigmoid(prediction)
57
58         y_trues.append(int(label[0]))
59         y_preds.append(probas[0].item())
60
61         try:
62             auc =
metrics.roc_auc_score(y_trues, y_preds)
63 -             accuracy =
metrics.accuracy_score(y_trues,
(np.array(y_preds) > 0.5).astype(int))-
64 -             sensitivity =
metrics.recall_score(y_trues,
(np.array(y_preds) > 0.5).astype(int))-
65 -             specificity =
metrics.recall_score(1 - np.array(y_trues), 1
- (np.array(y_preds) > 0.5).astype(int))-
66         except:
67             auc = 0.5
68 -             accuracy = 0.5-
69 -             sensitivity = 0.5-
70 -             specificity = 0.5-
71
72         writer.add_scalar('Train/Loss',
loss_value,
73                             epoch *
len(train_loader) + i)
74         writer.add_scalar('Train/AUC', auc,
epoch * len(train_loader) + i)
75
76         if (i % log_every == 0) & (i > 0):
77 -             print(
78                 '''[Epoch: {0} / {1} | Single
batch number : {2} / {3} ]| avg train loss
{4} | train auc : {5} | lr : {6}'''.
79                 format(
80                     epoch + 1,
81                     num_epochs,
82                     i,
83                     len(train_loader),
84                     np.round(np.mean(losses),
4),
85                     np.round(auc, 4),
86                     current_lr
87                 )
88             )
89
90         writer.add_scalar('Train/AUC_epoch', auc,
epoch)
91
92         train_loss_epoch =
np.round(np.mean(losses), 4)
93         train_auc_epoch = np.round(auc, 4)
94         train_accuracy_epoch = np.round(accuracy,
4)
95         train_sensitivity_epoch =
np.round(sensitivity, 4)
96 -         train_specificity_epoch =
np.round(specificity, 4)
97
98
99 -         return train_loss_epoch, train_auc_epoch,
train_accuracy_epoch,
train_sensitivity_epoch,
train_specificity_epoch
100 -

```

B oct-11-train.py

```

53         probas = torch.sigmoid(prediction)
54
55         y_trues.append(int(label[0]))
56         y_preds.append(probas[0].item())
57
58         try:
59             auc =
metrics.roc_auc_score(y_trues, y_preds)
60         except:
61             auc = 0.5
62
63         writer.add_scalar('Train/Loss',
loss_value,
64                             epoch *
len(train_loader) + i)
65         writer.add_scalar('Train/AUC', auc,
epoch * len(train_loader) + i)
66
67         if (i % log_every == 0) & (i > 0):
68             print('''[Epoch: {0} / {1} |
Single batch number : {2} / {3} ]| avg train
loss {4} | train auc : {5} | lr : {6}'''.
69             format(
70                 epoch + 1,
71                 num_epochs,
72                 i,
73                 len(train_loader),
74                 np.round(np.mean(losses), 4),
75                 np.round(auc, 4),
76                 current_lr
77             )
78
79         writer.add_scalar('Train/AUC_epoch', auc,
epoch)
80
81         train_loss_epoch =
np.round(np.mean(losses), 4)
82         train_auc_epoch = np.round(auc, 4)
83         train_accuracy_epoch = np.round(accuracy,
4)
84         train_sensitivity_epoch =
np.round(sensitivity, 4)
85         train_specificity_epoch =
np.round(specificity, 4)
86         return train_loss_epoch, train_auc_epoch,
train_accuracy_epoch,
train_sensitivity_epoch,
train_specificity_epoch
87

```

A train.py

```

101 -
102 def evaluate_model(model, val_loader, epoch,
    num_epochs, writer, current_lr, device,
    log_every=20, return_predictions=False):
103     """
104     Procedure to evaluate a model on the
    validation set
105     """
106     model.eval()
107
108     y_trues = []
109     y_preds = []
110     y_class_preds = []
111 -     adv_y_trues = []
112 -     adv_y_preds = []
113 -     adv_y_class_preds = []
114     losses = []
115 -     adv_losses = []
116 -     percent = args.advtrain_percent
117 -
118 -     # running adv validation
119 -     for i, (image, label, weight) in
    enumerate(val_loader):
120 -         stop_on = int(1130 * percent)
121 -         if i > stop_on:
122 -             break
123 -         image = image.to(device)
124 -         label = label.to(device)
125 -         weight = weight.to(device)
126 -         epsilon = args.epsilon
127 -         adv_image = fgsm_attack(model,
    F.binary_cross_entropy_with_logits, image,
    label, weight, epsilon, device="mps")
128 -         adv_prediction =
    model.forward(adv_image.float())
129 -         adv_loss =
    F.binary_cross_entropy_with_logits(adv_predic
    tion, label, weight=weight)
130 -         adv_loss_value = adv_loss.item()
131 -         adv_losses.append(adv_loss_value)
132 -         adv_probab =
    torch.sigmoid(adv_prediction)
133 -         adv_y_trues.append(int(label[0]))
134 -
    adv_y_preds.append(adv_probab[0].item())
135 -
    adv_y_class_preds.append((adv_probab[0] >
    0.5).float().item())
136 -
137 -     try:
138 -         adv_auc =
    metrics.roc_auc_score(adv_y_trues,
    adv_y_preds)
139 -     except:
140 -         adv_auc = 0.5
141 -
142 -     writer.add_scalar('Adv Val/Loss',
    adv_loss_value, epoch * len(val_loader) + i)
143 -     writer.add_scalar('Adv Val/AUC', adv_auc,
    epoch * len(val_loader) + i)
144 -     writer.add_scalar('Adv Val/AUC_epoch',
    adv_auc, epoch)
145 -     print('Adv Val/AUC_epoch', adv_auc,
    epoch)
146 -     val_adv_loss_epoch =
    np.round(np.mean(adv_losses), 4)

```

B oct-11-train.py

```

88 def evaluate_model(model, val_loader, epoch,
    num_epochs, writer, current_lr, device,
    log_every=20):
89     """
90     Procedure to evaluate a model on the
    validation set
91     """
92     model.eval()
93
94     y_trues = []
95     y_preds = []
96     y_class_preds = []
97
    losses = []

```

A train.py

```

147 -     val_adv_auc_epoch = np.round(adv_auc, 4)
148 -     try:
149 -         print("adv_y_trues: ", adv_y_trues,
150 -               "adv_y_class_preds: ", adv_y_class_preds)
151 -         val_adv_accuracy,
152 -         val_adv_sensitivity, val_adv_specificity =
153 -         ut.accuracy_sensitivity_specificity(adv_y_trues,
154 -         adv_y_class_preds)
155 -         val_adv_accuracy =
156 -         np.round(val_adv_accuracy, 4)
157 -         val_adv_sensitivity =
158 -         np.round(val_adv_sensitivity, 4)
159 -         val_adv_specificity =
160 -         np.round(val_adv_specificity, 4)
161 -     except:
162 -         val_adv_accuracy = 0.5
163 -         val_adv_sensitivity = 0.5
164 -         val_adv_specificity = 0.5
165
166     for i, (image, label, weight) in
167     enumerate(val_loader):
168
169         image = image.to(device)
170         label = label.to(device)
171         weight = weight.to(device)
172
173         prediction =
174         model.forward(image.float())
175
176         loss =
177         F.binary_cross_entropy_with_logits(prediction
178         , label, weight=weight)
179
180         loss_value = loss.item()
181         losses.append(loss_value)
182
183         probas = torch.sigmoid(prediction)
184
185         y_trues.append(int(label[0]))
186         y_preds.append(probas[0].item())
187         y_class_preds.append((probas[0] >
188         0.5).float().item())
189
190         try:
191             auc =
192             metrics.roc_auc_score(y_trues, y_preds)
193         except:
194             auc = 0.5
195
196         writer.add_scalar('Val/Loss',
197         loss_value, epoch * len(val_loader) + i)
198         writer.add_scalar('Val/AUC', auc,
199         epoch * len(val_loader) + i)
200
201         if (i % log_every == 0) & (i > 0):
202             print(
203                 '[Epoch: {0} / {1} | Single
204                 batch number : {2} / {3} ] | avg val loss {4}
205                 | val auc : {5} | lr : {6}'].
206                 format(
207                     epoch + 1,
208                     num_epochs,
209                     i,
210                     len(val_loader),
211                     np.round(np.mean(losses),
212                     4),

```

B oct-11-train.py

```

98
99     for i, (image, label, weight) in
100     enumerate(val_loader):
101
102         image = image.to(device)
103         label = label.to(device)
104         weight = weight.to(device)
105
106         prediction =
107         model.forward(image.float())
108
109         loss =
110         F.binary_cross_entropy_with_logits(prediction
111         , label, weight=weight)
112
113         loss_value = loss.item()
114         losses.append(loss_value)
115
116         probas = torch.sigmoid(prediction)
117
118         y_trues.append(int(label[0]))
119         y_preds.append(probas[0].item())
120         y_class_preds.append((probas[0] >
121         0.5).float().item())
122
123         try:
124             auc =
125             metrics.roc_auc_score(y_trues, y_preds)
126         except:
127             auc = 0.5
128
129         writer.add_scalar('Val/Loss',
130         loss_value, epoch * len(val_loader) + i)
131         writer.add_scalar('Val/AUC', auc,
132         epoch * len(val_loader) + i)
133
134         if (i % log_every == 0) & (i > 0):
135             print('[Epoch: {0} / {1} |
136             Single batch number : {2} / {3} ] | avg val
137             loss {4} | val auc : {5} | lr : {6}'].
138             format(
139                 epoch + 1,
140                 num_epochs,
141                 i,
142                 len(val_loader),
143                 np.round(np.mean(losses), 4),

```

A train.py

```

195         np.round(auc, 4),
196         current_lr
197     )
198 )
199
200     writer.add_scalar('Val/AUC_epoch', auc,
201                       epoch)
202
203     print('Val AUC: {}'.format(auc))
204
205     val_loss_epoch =
206     np.round(np.mean(losses), 4)
207     val_auc_epoch = np.round(auc, 4)
208
209     val_accuracy, val_sensitivity,
210     val_specificity =
211     ut.accuracy_sensitivity_specificity(y_trues,
212     y_class_preds)
213     val_accuracy = np.round(val_accuracy, 4)
214     val_sensitivity =
215     np.round(val_sensitivity, 4)
216     val_specificity =
217     np.round(val_specificity, 4)
218
219     if return_predictions:
220         return val_loss_epoch, val_auc_epoch,
221         val_accuracy, val_sensitivity,
222         val_specificity, y_preds, y_trues,
223         val_adv_loss_epoch, val_adv_auc_epoch,
224         val_adv_accuracy, val_adv_sensitivity,
225         val_adv_specificity, adv_y_preds, adv_y_trues
226     else:
227         return val_loss_epoch, val_auc_epoch,
228         val_accuracy, val_sensitivity,
229         val_specificity
230
231     '''
232
233     if return_predictions:
234         return val_loss_epoch, val_auc_epoch,
235         val_accuracy, val_sensitivity,
236         val_specificity, y_preds, y_trues
237     else:
238         return val_loss_epoch, val_auc_epoch,
239         val_accuracy, val_sensitivity,
240         val_specificity
241
242     '''
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

B oct-11-train.py

```

134         np.round(auc, 4),
135         current_lr
136     )
137 )
138
139     writer.add_scalar('Val/AUC_epoch', auc,
140                       epoch)
141
142     val_loss_epoch =
143     np.round(np.mean(losses), 4)
144     val_auc_epoch = np.round(auc, 4)
145
146     val_accuracy, val_sensitivity,
147     val_specificity =
148     ut.accuracy_sensitivity_specificity(y_trues,
149     y_class_preds)
150     val_accuracy = np.round(val_accuracy, 4)
151     val_sensitivity =
152     np.round(val_sensitivity, 4)
153     val_specificity =
154     np.round(val_specificity, 4)
155
156     if return_predictions:
157         return val_loss_epoch, val_auc_epoch,
158         val_accuracy, val_sensitivity,
159         val_specificity, y_preds, y_trues,
160         val_adv_loss_epoch, val_adv_auc_epoch,
161         val_adv_accuracy, val_adv_sensitivity,
162         val_adv_specificity, adv_y_preds, adv_y_trues
163     else:
164         return val_loss_epoch, val_auc_epoch,
165         val_accuracy, val_sensitivity,
166         val_specificity
167
168     '''
169
170     if return_predictions:
171         return val_loss_epoch, val_auc_epoch,
172         val_accuracy, val_sensitivity,
173         val_specificity, y_preds, y_trues
174     else:
175         return val_loss_epoch, val_auc_epoch,
176         val_accuracy, val_sensitivity,
177         val_specificity
178
179     '''
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

A train.py

```

237 -     adversarial_loss.backward()↵
238 - ↵
239 -     attack_image = image + eps *
240 -     image.grad.sign()↵
241 -     attack_image = torch.clamp(attack_image,
242 -     0, 1).detach()↵
243 -     return attack_image↵
244 - ↵
245 - def train_model_adv(model, epsilon,
246 -     train_loader, epoch, num_epochs, optimizer,
247 -     writer, current_lr, device, log_every,↵
248 -     retrain_percentage):↵
249 -     """↵
250 -     Procedure to train a model on the
251 -     training set by adversarial training↵
252 -     Method: FGSM↵
253 -     """↵
254 -     model.train()↵
255 -     ↵
256 -     model = model.to(device)↵
257 -     ↵
258 -     y_preds = []↵
259 -     y_trues = []↵
260 -     losses = []↵
261 -     ↵
262 -     train_times = int(retrain_percentage *
263 -     len(train_loader))↵
264 -     ↵
265 -     for i, (image, label, weight) in
266 -     enumerate(train_loader):↵
267 -     ↵
268 -         if i == train_times:↵
269 -             break↵
270 -         ↵
271 -         image = image.to(device)↵
272 -         label = label.to(device)↵
273 -         weight = weight.to(device)↵
274 -         ↵
275 -         # adversarial perturbation↵
276 -         if epsilon > 0:↵
277 -             adv_image = fgsm_attack(model,
278 -             F.binary_cross_entropy_with_logits, image,
279 -             label, weight, epsilon, device)↵
280 -         else:↵
281 -             adv_image = image↵
282 -         ↵
283 -         # adversarial training with perturbed
284 -         image↵
285 -         prediction = model(adv_image.float())↵
286 -         ↵
287 -         adv_loss =
288 -         F.binary_cross_entropy_with_logits(prediction
289 -         , label, weight=weight)↵
290 -         ↵
291 -         optimizer.zero_grad()↵
292 -         adv_loss.backward()↵
293 -         optimizer.step()↵
294 -         ↵
295 -         loss_value = adv_loss.item()↵
296 -         losses.append(loss_value)↵
297 -         ↵
298 -         probas = torch.sigmoid(prediction)↵
299 -         ↵
300 -         y_trues.append(int(label[0]))↵
301 -         y_preds.append(probas[0].item())↵

```

B oct-11-train.py

154

```

291 - 
292 - 
293 -     #
    torchviz.make_dot(prediction.mean(),
    params=dict(model.named_parameters())).render
    ("prediction", format="png")
294 - 
295 -     try:
296 -         auc =
    metrics.roc_auc_score(y_trues, y_preds)
297 -         accuracy =
    metrics.accuracy_score(y_trues,
    (np.array(y_preds) > 0.5).astype(int))
298 -         sensitivity =
    metrics.recall_score(y_trues,
    (np.array(y_preds) > 0.5).astype(int))
299 -         specificity =
    metrics.recall_score(1 - np.array(y_trues), 1
    - (np.array(y_preds) > 0.5).astype(int))
300 -     except:
301 -         auc = 0.5
302 -         accuracy = 0.5
303 -         sensitivity = 0.5
304 -         specificity = 0.5
305 - 
306 -     writer.add_scalar('Train/Loss',
    loss_value, epoch * train_times + i)
307 -     writer.add_scalar('Train/AUC', auc,
    epoch * train_times + i)
308 - 
309 -     if (i % log_every == 0) & (i > 0):
310 -         print(
311 -             '''[Epoch: {0} / {1} | Single
    batch number : {2} / {3} ] | avg train loss
    {4} | train auc : {5} | lr : {6} | eps:
    {7}'''.
312 -             format(
313 -                 epoch + 1,
314 -                 num_epochs,
315 -                 i,
316 -                 train_times,
317 -                 np.round(np.mean(losses),
    4),
318 -                 np.round(auc, 4),
319 -                 current_lr,
320 -                 epsilon
321 -             )
322 -         )
323 - 
324 -     print("===== adv train
    =====")
325 -     print("train times: ", train_times)
326 -     print("y_trues: ", y_trues, "y_preds: ",
    y_preds)
327 - 
328 -     writer.add_scalar('Train/AUC_epoch', auc,
    epoch)
329 - 
330 -     train_loss_epoch =
    np.round(np.mean(losses), 4)
331 -     train_auc_epoch = np.round(auc, 4)
332 -     train_accuracy_epoch = np.round(accuracy,
    4)
333 -     train_sensitivity_epoch =
    np.round(sensitivity, 4)
334 -     train_specificity_epoch =

```


A train.py

```

np.round(specificity, 4)~
335 - ~
336 -     return train_loss_epoch, train_auc_epoch,
    train_accuracy_epoch,
    train_sensitivity_epoch,
    train_specificity_epoch~
337 - ~
338 - ~
339 def run(args):
340     random.seed(args.seed)
341     torch.manual_seed(args.seed)
342     np.random.seed(args.seed)
343     torch.cuda.manual_seed_all(args.seed)
344
345     # create dirs to store experiment
    checkpoints, logs, and results
346     exp_dir_name = args.experiment
347     exp_dir = os.path.join('experiments',
    exp_dir_name)
348     if not os.path.exists(exp_dir):
349         os.makedirs(exp_dir)
350         os.makedirs(os.path.join(exp_dir,
    'models'))
351         os.makedirs(os.path.join(exp_dir,
    'logs'))
352         os.makedirs(os.path.join(exp_dir,
    'results'))
353
354     log_root_folder = exp_dir + "/logs/{0}/
    {1}/".format(args.task, args.plane)
355     if args.flush_history == 1:
356         objects = os.listdir(log_root_folder)
357         for f in objects:
358             if os.path.isdir(log_root_folder
    + f):
359                 shutil.rmtree(log_root_folder
    + f)
360
361     now = datetime.now()
362     logdir = log_root_folder +
    now.strftime("%Y%m%d-%H%M%S") + "/"
363     os.makedirs(logdir)
364
365     writer = SummaryWriter(logdir)
366
367     # create training and validation set
368     train_dataset = MRDataset(args.data_path,
    args.task, args.plane, train=True)
369     train_loader =
    torch.utils.data.DataLoader(train_dataset,
    batch_size=1, shuffle=True, num_workers=4,~
370 - drop_last=False)~
371
372     validation_dataset =
    MRDataset(args.data_path, args.task,
    args.plane, train=False)
373     validation_loader =
    torch.utils.data.DataLoader(validation_datase
    t, batch_size=1, shuffle=True,
    num_workers=2,~
374 - drop_last=False)~
375
376     if torch.cuda.is_available():

```

B oct-11-train.py

```

155 def run(args):
156     random.seed(args.seed)
157     torch.manual_seed(args.seed)
158     np.random.seed(args.seed)
159     torch.cuda.manual_seed_all(args.seed)
160
161     # create dirs to store experiment
    checkpoints, logs, and results
162     exp_dir_name = args.experiment
163     exp_dir = os.path.join('experiments',
    exp_dir_name)
164     if not os.path.exists(exp_dir):
165         os.makedirs(exp_dir)
166         os.makedirs(os.path.join(exp_dir,
    'models'))
167         os.makedirs(os.path.join(exp_dir,
    'logs'))
168         os.makedirs(os.path.join(exp_dir,
    'results'))
169
170     log_root_folder = exp_dir + "/logs/{0}/
    {1}/".format(args.task, args.plane)
171     if args.flush_history == 1:
172         objects = os.listdir(log_root_folder)
173         for f in objects:
174             if os.path.isdir(log_root_folder
    + f):
175                 shutil.rmtree(log_root_folder
    + f)
176
177     now = datetime.now()
178     logdir = log_root_folder +
    now.strftime("%Y%m%d-%H%M%S") + "/"
179     os.makedirs(logdir)
180
181     writer = SummaryWriter(logdir)
182
183     # create training and validation set
184     train_dataset = MRDataset(args.data_path,
    args.task, args.plane, train=True)
185     train_loader =
    torch.utils.data.DataLoader(train_dataset,
    batch_size=1, shuffle=True, num_workers=4,
    drop_last=False)~
186
187     validation_dataset =
    MRDataset(args.data_path, args.task,
    args.plane, train=False)
188     validation_loader =
    torch.utils.data.DataLoader(validation_datase
    t, batch_size=1, shuffle=True,
    num_workers=2, drop_last=False)~
189
190     if torch.cuda.is_available():

```


A train.py

```

377         device = torch.device('cuda')
378     else:
379         device = torch.device('mps')
380
381     # create the model
382     mrnet = MRNet()
383     mrnet = mrnet.to(device)
384
385     if args.advtrain == 1:
386         weights_name = f'./experiments/
baseline/models/model_{args.prefix_name}
_{args.task}_{args.plane}.pth'
387         print("[INFO] Loading weights:",
weights_name)
388
389         model = torch.load(weights_name)
390         model = model.to(device)
391         # load weights
392         model.eval()
393
394         if args.advtrain == 1:
395             optimizer =
optim.Adam(model.parameters(), lr=args.lr,
weight_decay=0.01)
396         else:
397             optimizer =
optim.Adam(mrnet.parameters(), lr=args.lr,
weight_decay=0.01)
398
399         if args.lr_scheduler == "plateau":
400             scheduler =
torch.optim.lr_scheduler.ReduceLROnPlateau(
optimizer, patience=5, factor=.3,
401 threshold=1e-4, verbose=True)
402         elif args.lr_scheduler == "step":
403             scheduler =
torch.optim.lr_scheduler.StepLR(
404                 optimizer, step_size=3,
gamma=args.gamma)
405
406         best_val_loss = float('inf')
407         best_val_auc = float(0)
408         best_val_accuracy = float(0)
409         best_val_sensitivity = float(0)
410         best_val_specificity = float(0)
411
412         num_epochs = args.epochs
413         iteration_change_loss = 0
414         patience = args.patience
415         log_every = args.log_every
416
417         t_start_training = time.time()
418         all_preds = []
419         all_labels = []
420
421         # train and test loop
422         for epoch in range(num_epochs):
423             current_lr = get_lr(optimizer)
424
425             t_start = time.time()
426
427             # train
428             if args.advtrain == 1:
429                 train_loss, train_auc,

```

B oct-11-train.py

```

191         device = torch.device('cuda')
192     else:
193         device = torch.device('cpu')
194
195     # create the model
196     mrnet = MRNet()
197     mrnet = mrnet.to(device)
198
199     optimizer =
optim.Adam(mrnet.parameters(), lr=args.lr,
weight_decay=0.01)
200
201     if args.lr_scheduler == "plateau":
202         scheduler =
torch.optim.lr_scheduler.ReduceLROnPlateau(
optimizer, patience=5, factor=.3,
203 threshold=1e-4, verbose=True)
204     elif args.lr_scheduler == "step":
205         scheduler =
torch.optim.lr_scheduler.StepLR(
206             optimizer, step_size=3,
gamma=args.gamma)
207
208     best_val_loss = float('inf')
209     best_val_auc = float(0)
210     best_val_accuracy = float(0)
211     best_val_sensitivity = float(0)
212     best_val_specificity = float(0)
213
214     num_epochs = args.epochs
215     iteration_change_loss = 0
216     patience = args.patience
217     log_every = args.log_every
218
219     t_start_training = time.time()
220
221     # train and test loop
222     for epoch in range(num_epochs):
223         current_lr = get_lr(optimizer)

```

```

train_accuracy, train_sensitivity,
train_specificity = train_model_adv(model,
args.epsilon, train_loader, epoch,
num_epochs, optimizer,
430 -
writer, current_lr, device, log_every,
args.advtrain_percent)
431 -
val_loss, val_auc, val_accuracy,
val_sensitivity, val_specificity, val_preds,
val_labels, val_adv_loss_epoch,
val_adv_auc_epoch, val_adv_accuracy,
val_adv_sensitivity, val_adv_specificity,
adv_y_preds, adv_y_trues =
evaluate_model(model, validation_loader,
epoch, num_epochs, writer, current_lr,
device, return_predictions=True)
432 -
else:
433 -
train_loss, train_auc,
train_accuracy, train_sensitivity,
train_specificity = train_model(mrnet,
train_loader, epoch, num_epochs, optimizer,
writer, current_lr,
434 -
device, log_every)
435 -
val_loss, val_auc, val_accuracy,
val_sensitivity, val_specificity, val_preds,
val_labels = evaluate_model(mrnet,
validation_loader, epoch, num_epochs, writer,
current_lr, device, return_predictions=True)
436 -
# calculate samples [find error]
437 -
438 -
439 -
all_preds.extend(val_preds)
440 -
all_labels.extend(val_labels)
441 -
442 -
# all_adv_preds.extend(adv_y_preds)
443 -
# all_adv_labels.extend(adv_y_trues)
444 -
445 -
if args.lr_scheduler == 'plateau':
446 -
scheduler.step(val_loss)
447 -
elif args.lr_scheduler == 'step':
448 -
scheduler.step()
449 -
450 -
t_end = time.time()
451 -
delta = t_end - t_start
452 -
453 -
learning_curve_csv =
f'learning_curve_{args.prefix_name}
_{args.task}_{args.plane}.csv'
454 -
455 -
filename = os.path.join(exp_dir,
'results', learning_curve_csv)
456 -
# Check if file exists
457 -
if os.path.exists(filename):
458 -
mode = 'a'
459 -
else:
460 -
mode = 'w'
461 -
462 -
# Open file and append or write to it
463 -
with open(filename, mode) as
res_file:
464 -
fa = csv.writer(res_file,
delimiter=',', quotechar='|',
quoting=csv.QUOTE_MINIMAL)
465 -
if mode == 'w':
466 -
# write headers if the file

```

```

is newly created~
467 -         fa.writerow(['epoch',
'train_loss', 'train_auc', 'train_accuracy',
'train_sensitivity', 'train_specificity',
'val_loss', 'val_auc', 'val_accuracy',
'val_sensitivity', 'val_specificity',
'val_adv_loss', 'val_adv_auc',
'val_adv_accuracy', 'val_adv_sensitivity',
'val_adv_specificity'])~
468 -         fa.writerow([epoch, train_loss,
train_auc, train_accuracy, train_sensitivity,
train_specificity, val_loss, val_auc,
val_accuracy, val_sensitivity,
val_specificity, val_adv_loss_epoch,
val_adv_auc_epoch, val_adv_accuracy,
val_adv_sensitivity, val_adv_specificity])~
469 - ~
470 - ~
471 -         print("train loss : {0} | train auc
{1} | val loss {2} | val auc {3} | elapsed
time {4} s".format(~
472 -             train_loss, train_auc, val_loss,
val_auc, delta))~
473 - ~
474 -             iteration_change_loss += 1~
475 -             print('-' * 30)~
476 - ~
477 -             if val_auc > best_val_auc:~
478 -                 best_val_auc = val_auc~
479 -                 best_val_accuracy = val_accuracy~
480 -                 best_val_sensitivity =
val_sensitivity~
481 -                 best_val_specificity =
val_specificity~
482 -                 if bool(args.save_model):~
483 -                     file_name =
f'model_{args.prefix_name}_{args.task}
_{args.plane}.pth'~
484 -                     for f in os.listdir(exp_dir +
'/models/'):~
485 -                         if (args.task in f) and
(args.plane in f) and (args.prefix_name in
f):~
486 -                             os.remove(exp_dir +
f'/models/{f}')~
487 -                             if args.advtrain == 1:~
488 -                                 torch.save(model, exp_dir
+ f'/models/{file_name}')~
489 -                             else:~
490 -                                 torch.save(mrnet, exp_dir
+ f'/models/{file_name}')~
491 - ~
492 -                     if val_loss < best_val_loss:~
493 -                         best_val_loss = val_loss~
494 -                         iteration_change_loss = 0~
495 - ~
496 -                     if iteration_change_loss == patience:~
497 -                         print('Early stopping after {0}
iterations without the decrease of the val
loss'.~
498 -                             format(iteration_change_loss))~
499 -                         break~
500 - ~
501 -             # save results to csv file~
502 -             with open(os.path.join(exp_dir,

```

```

    'results', f'model_{args.prefix_name}
    _{args.task}_{args.plane}-results.csv'),
503 -         'w') as res_file:
504 -         fw = csv.writer(res_file,
    delimiter=',', quotechar='|',
    quoting=csv.QUOTE_MINIMAL)
505 -         fw.writerow(['LOSS', 'AUC-best',
    'Accuracy-best', 'Sensitivity-best',
    'Specifity-best'])
506 -         fw.writerow([best_val_loss,
    best_val_auc, best_val_accuracy,
    best_val_sensitivity, best_val_specificity])
507 -         res_file.close()
508 -
509 -
510 -     # draw ROC curve for best model on
    validation set
511 -     fpr = []
512 -     tpr = []
513 -     # all_labels are validation ground truth
    labels
514 -     # all_preds are validation predictions
    from the model
515 -     fpr, tpr, thresholds =
    roc_curve(all_labels, all_preds)
516 -     print(fpr, tpr)
517 -     filename = "roc_curve_" +
    args.prefix_name + "_" + args.task + "_" +
    args.plane
518 -     with open(os.path.join(exp_dir,
    'results', filename), 'w') as f:
519 -         writer = csv.writer(f)
520 -         writer.writerow(['FPR', 'TPR',
    'Threshold'])
521 -         for i in range(len(fpr)):
522 -             writer.writerow([fpr[i], tpr[i],
    thresholds[i]])
523 -
524 -     t_end_training = time.time()
525 -     print(f'training took {t_end_training -
    t_start_training} s')
526 -
527 -
528 - def parse_arguments():
529 -     parser = argparse.ArgumentParser()
530 -     parser.add_argument('-t', '--task',
    type=str, required=True,
    choices=['abnormal',
531 - 'acl', 'meniscus'])
532 -     parser.add_argument('-p', '--plane',
    type=str, required=True,
    choices=['sagittal',
533 - 'coronal', 'axial'])
534 -     parser.add_argument('--data-path',
    type=str)
535 -     parser.add_argument('--prefix_name',
    type=str, required=True)
536 -     parser.add_argument('--experiment',
    type=str, required=True)
537 -     parser.add_argument('--augment',
    type=int, choices=[0, 1], default=1)
538 -     parser.add_argument('--lr_scheduler',
    type=str,
    default='plateau',
    choices=['plateau', 'step'])
539 -     parser.add_argument('--seed', type=int,

```

A train.py

```
        default=42)↵
541 -     parser.add_argument('--gamma',
        type=float, default=0.5)↵
542 -     parser.add_argument('--epochs', type=int,
        default=50)↵
543 -     parser.add_argument('--lr', type=float,
        default=1e-6)↵
544 -     parser.add_argument('--flush_history',
        type=int, choices=[0, 1], default=0)↵
545 -     parser.add_argument('--save_model',
        type=int, choices=[0, 1], default=1)↵
546 -     parser.add_argument('--patience',
        type=int, default=50)↵
547 -     parser.add_argument('--log_every',
        type=int, default=100)↵
548 - ↵
549 -     # Adversarial training arguments↵
550 -     parser.add_argument('--advtrain',
        type=int, choices=[0, 1], default=0)↵
551 -     parser.add_argument('--advtrain_percent',
        type=float)↵
552 -     parser.add_argument('--epsilon',
        type=float)↵
553 - ↵
554 -     args = parser.parse_args()↵
555 -     return args↵
556 - ↵
557 - ↵
558 - if __name__ == "__main__":↵
559 -     args = parse_arguments()↵
560 -     run(args)↵
561
```

B oct-11-train.py

224