

# Homework 1

Monday, January 25, 2021

12:08 AM

## Problem 1: Backpropagation (25Pts)

Assume we have defined a loss function  $l$ , now for a given layer  $y = f(x, W)$ , write  $\frac{\partial l}{\partial x}$  and  $\frac{\partial l}{\partial W}$  as functions of  $\frac{\partial l}{\partial y}$ ,  $x$ ,  $y$ , and  $W$  in the following cases:

a)  $y = xW$ ,  $x \in \mathbb{R}^{1 \times n}$ ,  $W \in \mathbb{R}^{n \times m}$

$$\frac{\partial l}{\partial x} = \frac{\partial l}{\partial y} \frac{\partial y}{\partial x} = \frac{\partial l}{\partial y} W^T$$

$$\frac{\partial l}{\partial W} = \frac{\partial l}{\partial y} \frac{\partial y}{\partial W} = x^T \frac{\partial l}{\partial y}$$

b)  $y = xW$ ,  $x \in \mathbb{C}^{1 \times n}$ ,  $W \in \mathbb{C}^{n \times m}$

Paych reference on complex derivatives :

$$\frac{\partial l}{\partial x^*} = \left( \frac{\partial l}{\partial y} \right)^* \cdot \frac{\partial y}{\partial x^*} + \frac{\partial l}{\partial y} \left( \frac{\partial y}{\partial x^*} \right)^*$$

$$x = a + ib, \quad a, b \in \mathbb{R}^n$$

$$\frac{\partial y}{\partial x} = \frac{\partial (xW)}{\partial x} = W^T, \quad \frac{\partial y}{\partial x^*} = 0$$

$$\frac{\partial l}{\partial x} = \frac{\partial l}{\partial y^*} W^{*T}$$

$$\frac{\partial l}{\partial W} = x^T \frac{\partial l}{\partial y^*}$$

c)  $y = \|xW\|_2^2$ ,  $x \in \mathbb{R}^{1 \times n}$ ,  $W \in \mathbb{R}^{n \times m}$

$$\frac{\partial l}{\partial x} = \frac{\partial l}{\partial y} 2(xW)W^T$$

$$\frac{\partial l}{\partial W} = \frac{\partial l}{\partial y} 2x^T(xW)$$

d)  $y = x \odot w$ ,  $y_k = x_k w_k$ ,  $x$  and  $w \in \mathbb{R}^n$

$$\frac{\partial l}{\partial x} = \frac{\partial l}{\partial y} \odot w$$

$$\frac{\partial l}{\partial w} = \frac{\partial l}{\partial y} \odot x$$

e)  $y = \text{softmax}(x)$ ,  $y_k = \frac{e^{x_k}}{\sum_{i=0}^{n-1} e^{x_i}}$ ,  $x \in \mathbb{R}^n$

$$\frac{\partial l}{\partial x} = \frac{\partial l}{\partial y} \frac{\partial y}{\partial x} \quad \text{Split into case when } i=k \text{ \& } i \neq k$$

For  $i=k$ :

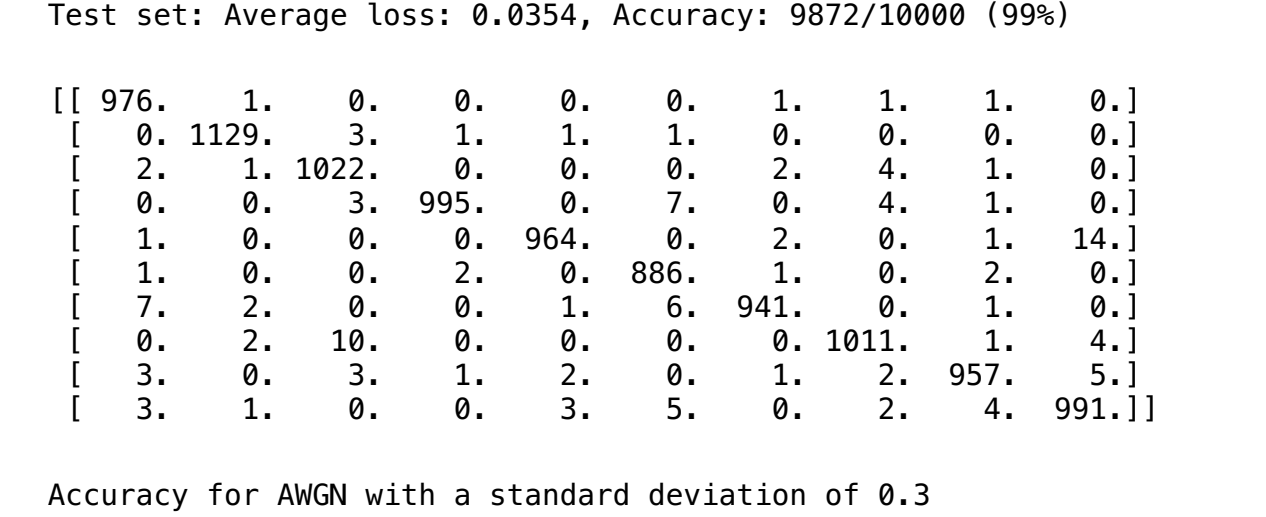
$$\frac{\partial z_k}{\partial x_i} = \frac{\partial}{\partial x_i} \left( \sum_{c=0}^{n-1} e^{x_c} \right) = e^{x_i}$$
$$\frac{\partial l}{\partial x_i} = \frac{\partial l}{\partial y_k} \frac{\partial y_k}{\partial x_i} = \frac{\partial l}{\partial y_k} \left( \frac{e^{x_k} - e^{x_k} \sum_{c \neq k} e^{x_c}}{( \sum_{c=0}^{n-1} e^{x_c} )^2} \right)$$
$$\frac{\partial l}{\partial x_i} = \frac{\partial l}{\partial y_k} \frac{\partial y_k}{\partial x_i} = \frac{\partial l}{\partial y_k} \left( \frac{e^{x_k} - e^{x_k} \sum_{c \neq k} e^{x_c}}{( \sum_{c=0}^{n-1} e^{x_c} )^2} \right)$$
$$= \frac{\partial l}{\partial y_k} \left( \frac{e^{x_k}}{\sum_{c=0}^{n-1} e^{x_c}} \left[ \frac{e^{x_k}}{\sum_{c=0}^{n-1} e^{x_c}} - 1 \right] \right)$$
$$= \frac{\partial l}{\partial y_k} \left( \text{softmax}(x_i) [1 - \text{softmax}(x_i)] \right)$$

For  $i \neq k$ :

$$\frac{\partial l}{\partial x_i} = \frac{\partial l}{\partial y_k} \frac{\partial y_k}{\partial x_i} = \frac{\partial l}{\partial y_k} \left( \frac{-e^{x_k} e^{x_i}}{( \sum_{c=0}^{n-1} e^{x_c} )^2} \right)$$
$$\frac{\partial l}{\partial x_i} = \frac{\partial l}{\partial y_k} \left( \frac{-e^{x_k} e^{x_i}}{( \sum_{c=0}^{n-1} e^{x_c} )^2} \right) = \frac{\partial l}{\partial y_k} \left( -\text{softmax}(x_k) \text{softmax}(x_i) \right)$$

Read through the sample script you downloaded: Train the model for 2 epochs, and save the weights. This should be done using command line arguments, without modifying the original file. Include the bash command you used in your HW submission.

Bash command: python3 main.py --epochs 2 --no-cuda --save-model



Accuracy for AWGN with a standard deviation of 0

Test set: Average loss: 0.0354, Accuracy: 9872/10000 (99%)

[ [ 976.	1.	0.	0.	0.	0.	1.	1.	1.	0.]
[ 0.	1129.	3.	1.	1.	1.	0.	0.	0.	0.]
[ 2.	1.	1022.	0.	0.	0.	2.	4.	1.	0.]
[ 0.	0.	0.	995.	0.	0.	7.	0.	4.	0.]
[ 1.	0.	0.	0.	964.	0.	2.	0.	1.	14.]
[ 1.	0.	0.	2.	0.	886.	1.	0.	2.	0.]
[ 7.	2.	0.	0.	1.	6.	941.	0.	1.	0.]
[ 0.	2.	10.	0.	0.	0.	0.	1011.	1.	4.]
[ 3.	0.	3.	1.	2.	0.	1.	2.	957.	5.]
[ 3.	1.	0.	0.	3.	5.	0.	2.	4.	991.]]

Accuracy for AWGN with a standard deviation of 0.3

Test set: Average loss: 0.2217, Accuracy: 9566/10000 (96%)

[ [ 959.	0.	8.	3.	0.	1.	2.	0.	6.	1.]
[ 0.	1049.	63.	1.	11.	2.	0.	5.	4.	0.]
[ 1.	1.	1026.	0.	0.	0.	1.	0.	3.	0.]
[ 0.	0.	15.	960.	1.	20.	0.	3.	9.	2.]
[ 1.	0.	9.	0.	959.	0.	1.	1.	5.	7.]
[ 1.	0.	0.	8.	0.	858.	5.	1.	17.	2.]
[ 4.	0.	10.	1.	3.	7.	914.	0.	19.	0.]
[ 1.	3.	59.	0.	2.	2.	0.	947.	4.	10.]
[ 1.	1.	10.	0.	0.	2.	1.	2.	956.	1.]
[ 2.	1.	2.	2.	21.	7.	1.	6.	29.	938.]]

Accuracy for AWGN with a standard deviation of 0.6

Test set: Average loss: 1.3472, Accuracy: 5098/10000 (51%)

[ [ 443.	0.	282.	14.	8.	29.	16.	0.	178.	0.]
[ 0.	128.	637.	23.	58.	37.	6.	29.	217.	0.]
[ 0.	1.	915.	10.	5.	3.	3.	87.	3.	3.]
[ 0.	1.	163.	524.	5.	63.	1.	5.	242.	6.]
[ 0.	0.	114.	10.	636.	17.	5.	5.	161.	34.]
[ 1.	0.	74.	45.	14.	428.	7.	1.	314.	8.]
[ 2.	2.	165.	2.	38.	43.	468.	3.	232.	3.]
[ 1.	5.	363.	37.	70.	48.	2.	375.	82.	45.]
[ 1.	0.	92.	12.	4.	5.	0.	1.	859.	0.]
[ 1.	0.	106.	22.	159.	28.	4.	20.	347.	322.]]

Accuracy for AWGN with a standard deviation of 1.0

Test set: Average loss: 2.2629, Accuracy: 2164/10000 (22%)

```

11
12
13 class ANNG(Object):
14     def __init__(self, sigma, vmin=0, vmax=1):
15         assert isinstance(sigma, (int, float))
16         self.sigma = sigma
17         self.vmin = vmin
18         self.vmax = vmax
19
20     def __call__(self, image):
21         noise = torch.from_numpy(np.random.normal(0.0, self.
22 astype(np.float32))
23         # Above supports std of 0
24
25         # noise = torch.normal(0.0, self.sigma, image.shape)
26         output = image + noise
27         torch.clip(output, self.vmin, self.vmax, out=output)
28

```

## Code:

The matplotlib code for showing the images with various AWGN and the computations for the confusion matrix are on main.py. Setting up and loading the model is in test\_AWGN.py.

```
File - Users\wwwelamarc\Google Drive\School\Machine Learning\ECE281 Computer Vision\Homeworks\Homework1\image_cnn
1 from __future__ import print_function
2 import argparse
3 import torch
4 import torch.nn as nn
5 import torch.nn.functional as F
6 import torch.optim as optim
7 import numpy as np
8 import matplotlib.pyplot as plt
9 from torchvision import datasets, transforms
10 from torch.optim.lr_scheduler import StepLR
11
12
13 class AWGN(object):
14     def __init__(self, sigma, vmin=0, vmax=1):
15         assert isinstance(sigma, (int, float))
16         self.sigma = sigma
17         self.vmin = vmin
18         self.vmax = vmax
19
20     def __call__(self, image):
21         noise = torch.from_numpy(np.random.normal(0, self.sigma, image.shape).
22                                 astype(np.float32))
23         # Above supports std of 0
24         # noise = torch.normal(0, self.sigma, image.shape)
25         output = image + noise
26         torch.clip(output, self.vmin, self.vmax, out=output)
27
28         return output
29
30
31 class Net(nn.Module):
32     def __init__(self):
33         super(Net, self).__init__()
34         self.conv2 = nn.Conv2d(1, 32, 3, 1)
35         self.conv2 = nn.Conv2d(32, 64, 3, 1)
36         self.dropout1 = nn.Dropout(0.25)
37         self.dropout2 = nn.Dropout(0.5)
38         self.fc1 = nn.Linear(9216, 128)
39         self.fc2 = nn.Linear(128, 10)
40
41     def forward(self, x):
42         x = self.conv2(x)
43         x = F.relu(x)
44         x = self.conv2(x)
45         x = F.relu(x)
46         x = F.max_pool2d(x, 2)
47         x = self.dropout1(x)
48         x = torch.flatten(x, 1)
49         x = self.fc1(x)
50         x = F.relu(x)
51         x = self.dropout2(x)
52         x = self.fc2(x)
53         output = F.log_softmax(x, dim=1)
54         return output
55
56 def train(args, model, device, train_loader, optimizer, epoch):
57     model.train()
58     for batch_idx, (data, target) in enumerate(train_loader):
59         data, target = data.to(device), target.to(device)
60         optimizer.zero_grad()
61         output = model(data)
62         loss = F.nll_loss(output, target)
63         loss.backward()
64         optimizer.step()
65         if batch_idx % args.log_interval == 0:
66             print('Train Epoch: {} [{}/{} ({:.0f)%}]\tloss: {:.4f}'.format(
67                 epoch, batch_idx * len(data), len(data_loader.dataset),
68                 100. * batch_idx / len(train_loader), loss.item()))
69             if args.dry_run:
70                 break
71
72
73 def test(model, device, test_loader):
74     model.eval()
75     test_loss = 0
76     correct = 0
77
78     confusion_matrix = np.zeros((10,10))
79     with torch.no_grad():
80         for data, target in test_loader:
81             data, target = data.to(device), target.to(device)
82             output = model(data)
83             test_loss += F.nll_loss(output, target, reduction='sum').item() # sum
84             # up batch loss
85             pred = output.argmax(dim=1, keepdim=True) # get the index of the max
86             # probability
87             indices = np.array([target.numpy().squeeze(), pred.numpy().squeeze()]).
88             transpose()
89             for i in indices:
90                 confusion_matrix[i[0],i[1]] += 1
91
92     correct += pred.eq(target.view_as(pred)).sum().item()
93
94     test_loss /= len(test_loader.dataset)
95
96     print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
97         test_loss, correct, len(test_loader.dataset),
98         100. * correct / len(test_loader.dataset)))
99
100 # suppress: suppress scientific notation
101 with np.printoptions(precision=1, suppress=True):
102     print(np.array(confusion_matrix))
103
104
105 def main():
106     # Training settings
107     parser = argparse.ArgumentParser(description='PyTorch MNIST Example')
108     parser.add_argument('--batch-size', type=int, default=64, metavar='N',
109                         help='input batch size for training (default: 64)')
110     parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
111                         help='input batch size for testing (default: 1000)')
112     parser.add_argument('--epochs', type=int, default=20, metavar='N',
113                         help='number of epochs to train (default: 20)')
114     parser.add_argument('--lr', type=float, default=0.01, metavar='LR',
115                         help='learning rate (default: 0.01)')
116     parser.add_argument('--gamma', type=float, default=0.7, metavar='M',
117                         help='learning rate step gamma (default: 0.7)')
118     parser.add_argument('--no-cuda', action='store_true', default=False,
119                         help='disables CUDA training')
120     parser.add_argument('--dry-run', action='store_true', default=False,
121                         help='quickly check a single pass')
122     parser.add_argument('--seed', type=int, default=1, metavar='S',
123                         help='random seed (default: 1)')
124     parser.add_argument('--log-interval', type=int, default=10, metavar='N',
125                         help='how many batches to wait before logging training
126                             status')
127     parser.add_argument('--save-model', action='store_true', default=False,
128                         help='if save model')
129     parser.add_argument('--test-AWGN', action='store_true', default=False,
130                         help='test AWGN transformation')
131     parser.add_argument('--sigma', type=float, default=0.0, metavar='N',
132                         help='standard deviation of AWGN')
133     args = parser.parse_args()
134     use_cuda = not args.no_cuda and torch.cuda.is_available()
135
136     torch.manual_seed(args.seed)
137
138     device = torch.device("cuda" if use_cuda else "cpu")
139
140     train_kwargs = {'batch_size': args.batch_size}
141     test_kwargs = {'batch_size': args.test_batch_size}
142
143     if use_cuda:
144         cuda_kwargs = {'num_workers': 1,
145                       'pin_memory': True,
146                       'shuffle': True}
147         train_kwargs.update(cuda_kwargs)
148         test_kwargs.update(cuda_kwargs)
149
150     if args.test_AWGN:
151         transform = transforms.Compose([transforms.ToTensor()])
152         test_dataset = datasets.MNIST('..data', train=False, transform=transform)
153
154         #PDEBERG STEP
155         print(type(test_dataset[0][0]))
156         print(test_dataset[0][0].shape)
157         print(type(test_dataset[0][0].numpy()))
158         print(test_dataset[0][0].numpy().shape)
159
160         # Display effects of AWGN on first image
161         fig = plt.figure()
162         image = test_dataset[0][0]
163         print(image.type())
164
165         for i, sigma in enumerate([0.001, 0.3, 0.6, 1.0]):
166             add_noise = AWGN(sigma)
167             transformed_sample = add_noise(image)
168             print(transformed_sample.type())
169
170         ax = plt.subplot(1, 4, 1 + 1)
171         plt.tight_layout()
172         plt.axis('off')
173
174         ax.set_title('%s\Sigma = %f' % (sigma, sigma))
175         plt.imshow(transformed_sample.numpy().squeeze(), cmap='gray')
176
177         fig.show()
178         plt.show()
179
180         #
181
182     #
183     # test_loader = torch.utils.data.DataLoader(test_dataset, **test_kwargs)
184     else:
185         transform = transforms.Compose([transforms.ToTensor(),
186                                         transforms.Normalize((0.1307,), (0.3081,))])
187         test_dataset = datasets.MNIST('..data', train=True, download=True,
188                                     transform=transform)
189         test_dataset = datasets.MNIST('..data', train=False, download=True,
190                                     transform=transform)
191         train_loader = torch.utils.data.DataLoader(test_dataset, **test_kwargs)
192         test_loader = torch.utils.data.DataLoader(test_dataset, **test_kwargs)
193
194         model = Net().to(device)
195         optimizer = optim.Adam(model.parameters(), lr=args.lr)
196
197         scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)
198         for epoch in range(1, args.epochs + 1):
199             train(args, model, device, train_loader, optimizer, epoch)
200             test(model, device, test_loader)
201             scheduler.step()
202
203             if args.save_model:
204                 torch.save(model.state_dict(), 'mnist_cnn.pt')
205
206         if __name__ == '__main__':
207             main()
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
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
10
```