CPSC 532W Homework 4

Naomi Graham

March 10, 2021

All the code can be found on: `https://github.com/n6graham/cpsc_hw4`.
Sadly, again, I ran out of time for plotting, but I really think my code works well in theory!

1 BBVI code

For BBVI, I used the graph-based implementation based off of Jason's HW2 code.

1.1 BBVI Algorithm

Even though I was getting good proposals from running SGD, it seems that a bug in my importance sampling was causing it to look like I had really bad values after sampling.

For example, In program 1, my variational posterior ended up being:

```
    sigma['Q'] is  {'sample2': Normal(loc: 7.149489402770996, scale:
0.9122980237007141)}

```

But the posterior mean and variance returned by importance sampling were:
Mean: 1.3713 Variance: 3.5675.

1.2 Code

I implemented SGD for optimizer-step. Also returned the value max-norm to keep track of how the norm of the gradient was looking. It was almost always somewhat-monotonically decreasing.

```
    def optimizer_step(q, ghat,t):
            for v,d in q.items():
                print("norm of gradient:", np.linalg.norm(ghat[v]))
                i = 0
                for params in d.Parameters():
                    params.data = params.data + ghat[v][i]/(t+10)

            max_norm = np.max([ np.linalg.norm(ghat[v]) for v,d in q.items()])

            return q,max_norm

```

I also implemented elbo-grad, which was a lot of work (but worth it, of course!)

```
    def elbo_grad(Glist,logWlist):
            L = len(Glist)
            Flist = list([{} for i in range(0,L)])
            ghat = {}
            U = list(set([u for G in Glist for u in G]))
            print("here!!!")

            for v in U:
                # get number of parameters for v
                for i in range(0,L):
                    if v in list(Glist[i].keys()):
                        num_params = len(Glist[i][v])
                        break

                for i in range(0,L):
                    if v in list(Glist[i].keys()):
                        x = Glist[i][v]*logWlist[i]
                        if i == (L-1): print("x is ", x)
                        Flist[i][v] = x
                    else:
                        Flist[i][v] = torch.tensor([0 for j in range(
num_params)])
                        Glist[i][v] = torch.tensor([0 for j in range(
num_params)])

                Fv = [ Flist[i][v] for i in range(0,L)]
                Gv = [ Glist[i][v] for i in range(0,L)]
                Fv = torch.stack(Fv)
                Gv = torch.stack(Gv)
                Fv = Fv.detach().numpy()

                varG = [ np.var(np.array(Gv[:,j])) for j in range(num_params)
]
                denom = sum(varG)

                C = np.array([ np.cov(Fv[:,j],Gv[:,j], rowvar=True) for j in
range(num_params) ])
                cov = [ C[j][1][0] for j in range(num_params) ]
                numerator = sum(cov)
                bhat = numerator/denom


                print("bhat is", bhat)

                #numerator = np.array([ np.sum(C[j]) for j in range(num_params
) ])


                ghat[v] = sum( np.divide((Fv - bhat*np.array(Gv)),L) )


            print("returning ghat:", ghat)

            return ghat

```

Here I implement algorithm 15.

```
    sigma = {'Q':{}, 'logW':0, 'G':{} }
    weighted_samples = []

    for t in range(0,T): # T is the number of iterations
        Glist = []
        logWlist = []

        # here we compute a batch of gradients
        for l in range(0,L): # L is the batch size
            sigma['logW']=0
            # first we get the trace and update sigma using sample from joint
            #val_l, sigma_l, trace_l = sample_from_joint(graph,sigma)
            r_l, sigma_l, trace_l = sample_from_joint(graph,sigma)
            # then we get the deterministic expression using the trace
            #deterministic_expr = plugin_parent_values(expr,trace_l)
            G_l = copy.deepcopy(sigma_l['G'])
            logW_l = sigma_l['logW']
            Glist.append(G_l)
            logWlist.append(logW_l)

            weighted_samples.append((r_l,logW_l))

        ELBO = sum(logWlist)

        ghat = elbo_grad(Glist,logWlist)

        sigma['Q'], max_norm = optimizer_step( sigma['Q'],ghat,t) #update the proposal
        print("results on iteration {} are ".format(t), sigma['Q'])
        print("the max gradient is ", max_norm )



    print("sigma['Q'] is ",sigma['Q'])

    return weighted_samples, sigma['Q']

```

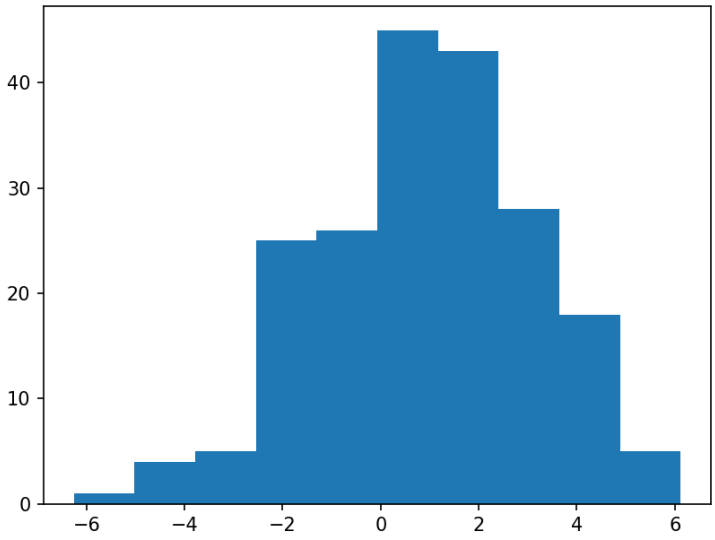
Sadly my importance sampling just was not working for some reason :(

2 Program 1

```
returning ghat: {'sample2': array([-0.49577966, -0.0261793 ], dtype=float32)}
norm of gradient: 0.49647036
results on iteration 93 are  {'sample2': Normal(loc: 7.7331366539001465, scale: 0.91798853874
20654)}
the max gradient is  0.49647036
ELBO is tensor(-951.8278, grad_fn=<AddBackward0>)
returning ghat: {'sample2': array([-0.5220273 ,  0.08419174], dtype=float32)}
norm of gradient: 0.5287729
results on iteration 94 are  {'sample2': Normal(loc: 7.728116989135742, scale: 0.917835891246
7957)}
the max gradient is  0.5287729
ELBO is tensor(-968.4595, grad_fn=<AddBackward0>)
returning ghat: {'sample2': array([-0.42646474, -0.03271492], dtype=float32)}
norm of gradient: 0.42771772
results on iteration 95 are  {'sample2': Normal(loc: 7.724055290222168, scale: 0.918322205543
5181)}
the max gradient is  0.42771772
ELBO is tensor(-951.6724, grad_fn=<AddBackward0>)
returning ghat: {'sample2': array([-0.39113718,  0.0641632 ], dtype=float32)}
norm of gradient: 0.39636502
results on iteration 96 are  {'sample2': Normal(loc: 7.720365524291992, scale: 0.918134987354
2786)}
the max gradient is  0.39636502
ELBO is tensor(-975.7202, grad_fn=<AddBackward0>)
returning ghat: {'sample2': array([-0.42080986,  0.08003196], dtype=float32)}
norm of gradient: 0.4283527
results on iteration 97 are  {'sample2': Normal(loc: 7.716432571411133, scale: 0.918498694896
698)}
the max gradient is  0.4283527
ELBO is tensor(-1007.1896, grad_fn=<AddBackward0>)
returning ghat: {'sample2': array([-0.38780287, -0.11757746], dtype=float32)}
norm of gradient: 0.40523517
results on iteration 98 are  {'sample2': Normal(loc: 7.712841987609863, scale: 0.918948173522
9492)}
the max gradient is  0.40523517
ELBO is tensor(-1002.4915, grad_fn=<AddBackward0>)
returning ghat: {'sample2': array([-0.44282976, -0.13143373], dtype=float32)}
norm of gradient: 0.46192318
results on iteration 99 are  {'sample2': Normal(loc: 7.708779335021973, scale: 0.918293952941
8945)}
the max gradient is  0.46192318
sigma['Q'] is  {'sample2': Normal(loc: 7.708779335021973, scale: 0.9182939529418945)}
{'sample2': Normal(loc: 7.708779335021973, scale: 0.9182939529418945)}
```

```
norm of gradient: 0.29127353
results on iteration 92 are  {'sample2': Normal(loc: 7.540454864501953, scale: 0.915859937667
8467)}
the max gradient is  0.29127353
ELBO is tensor(-1230.8153, grad_fn=<AddBackward0>)
norm of gradient: 0.31071016
results on iteration 93 are  {'sample2': Normal(loc: 7.537442684173584, scale: 0.915426969528
1982)}
the max gradient is  0.31071016
ELBO is tensor(-1245.2018, grad_fn=<AddBackward0>)
norm of gradient: 0.29013023
results on iteration 94 are  {'sample2': Normal(loc: 7.534736156463623, scale: 0.915524959564
209)}
the max gradient is  0.29013023
ELBO is tensor(-1246.6462, grad_fn=<AddBackward0>)
norm of gradient: 0.3307241
results on iteration 95 are  {'sample2': Normal(loc: 7.531586647033691, scale: 0.915119588375
0916)}
the max gradient is  0.3307241
ELBO is tensor(-1232.8281, grad_fn=<AddBackward0>)
norm of gradient: 0.26369676
results on iteration 96 are  {'sample2': Normal(loc: 7.529160499572754, scale: 0.915133357048
0347)}
the max gradient is  0.26369676
ELBO is tensor(-1250.5586, grad_fn=<AddBackward0>)
norm of gradient: 0.3192326
results on iteration 97 are  {'sample2': Normal(loc: 7.526177883148193, scale: 0.915463030338
2874)}
the max gradient is  0.3192326
ELBO is tensor(-1254.6959, grad_fn=<AddBackward0>)
norm of gradient: 0.26852316
results on iteration 98 are  {'sample2': Normal(loc: 7.523730754852295, scale: 0.915418386459
3506)}
the max gradient is  0.26852316
ELBO is tensor(-1268.3943, grad_fn=<AddBackward0>)
norm of gradient: 0.26799098
results on iteration 99 are  {'sample2': Normal(loc: 7.521484375, scale: 0.915682315826416)}
the max gradient is  0.26799098
sigma['Q'] is  {'sample2': Normal(loc: 7.521484375, scale: 0.915682315826416)}
proposal is: {'sample2': Normal(loc: 7.521484375, scale: 0.915682315826416)}
Mean:  tensor(0.0710, grad_fn=<SumBackward0>) Variance:  tensor(3.3861, grad_fn=<SubBackward0
>)
```

But the sampled values look wrong:



3 Program2

Again we can see that the optimized proposal is looking correct.

```
the max gradient is  103.12931
ELBO is tensor(7425.1748, grad_fn=<AddBackward0>)
norm of gradient: 50.927845
norm of gradient: 70.857544
results on iteration 195 are  {'sample1': Normal(loc: 2.3805956840515137, scale:
 0.1035614013671875), 'sample2': Normal(loc: 0.6931635737419128, scale: 8.503091
812133789)}
the max gradient is  70.857544
ELBO is tensor(13869.1396, grad_fn=<AddBackward0>)
norm of gradient: 117.126724
norm of gradient: 436.69858
results on iteration 196 are  {'sample1': Normal(loc: 2.3758506774902344, scale:
 0.10381706058979034), 'sample2': Normal(loc: 0.6936087608337402, scale: 8.50076
5800476074)}
the max gradient is  436.69858
ELBO is tensor(1138.0188, grad_fn=<AddBackward0>)
norm of gradient: 34.049946
norm of gradient: 58.89527
results on iteration 197 are  {'sample1': Normal(loc: 2.3748931884765625, scale:
 0.10417088121175766), 'sample2': Normal(loc: 0.6954396367073059, scale: 8.52292
537689209)}
the max gradient is  58.89527
ELBO is tensor(11153.1797, grad_fn=<AddBackward0>)
norm of gradient: 215.83969
norm of gradient: 197.07481
results on iteration 198 are  {'sample1': Normal(loc: 2.36855149269104, scale: 0
.10431227833032608), 'sample2': Normal(loc: 0.6991506814956665, scale: 8.5205812
45422363)}
the max gradient is  215.83969
ELBO is tensor(-1818.7682, grad_fn=<AddBackward0>)
norm of gradient: 175.42859
norm of gradient: 46.918015
results on iteration 199 are  {'sample1': Normal(loc: 2.36621356010437, scale: 0
.10518734157085419), 'sample2': Normal(loc: 0.7014965415000916, scale: 8.5297613
1439209)}
the max gradient is  175.42859
sigma['Q'] is  {'sample1': Normal(loc: 2.36621356010437, scale: 0.10518734157085
419), 'sample2': Normal(loc: 0.7014965415000916, scale: 8.52976131439209)}
proposal is: {'sample1': Normal(loc: 2.36621356010437, scale: 0.1051873415708541
9), 'sample2': Normal(loc: 0.7014965415000916, scale: 8.52976131439209)}
```

4 Program3

I have a bug that only affects program 3.

5 Program4

```
(loc: 0.028813054785132408, scale: 0.910309910774231), 'sample73': Normal(loc: -0.1274077445268631, scale: 1
211338043213), 'sample104': Normal(loc: 0.13367031514644623, scale: 1.000471591949463), 'sample9': Normal(lo
23851779103279114, scale: 0.863947868347168), 'sample121': Normal(loc: 0.12602509558200836, scale: 0.9069899
8865), 'sample102': Normal(loc: -0.020959576591849327, scale: 1.05450439453125), 'sample55': Normal(loc: -0.
716641902924, scale: 0.8955751657485962), 'sample87': Normal(loc: -0.0016292359214276075, scale: 0.821322619
85), 'sample8': Normal(loc: 0.15166804194450378, scale: 0.9865884184837341), 'sample51': Normal(loc: 0.02815
5573826, scale: 0.897223174571991), 'sample72': Normal(loc: 0.28447407484054565, scale: 1.041612148284912),
le39': Normal(loc: -0.23673585057258606, scale: 0.9712879657745361), 'sample7': Normal(loc: 0.26283746957778
cale: 0.8800844550132751), 'sample54': Normal(loc: -0.008857478387653828, scale: 0.9268837571144104), 'sampl
: Normal(loc: 0.14971597492694855, scale: 0.993392825126648), 'sample101': Normal(loc: 0.34996819496154785,
: 0.9475797414779663), 'sample61': Normal(loc: -0.03995126858353615, scale: 0.9674408435821533), 'sample115'
mal(loc: -0.12889261543750763, scale: 1.0636239051818848), 'sample17': Normal(loc: -0.14979583024978638, sca
.9294111132621765), 'sample131': Normal(loc: 0.3166612386703491, scale: 1.0845746994018555), 'sample107': No
loc: 0.15554308891296387, scale: 1.019728422164917), 'sample109': Normal(loc: 0.09929069131612778, scale: 0.
23301315308), 'sample25': Normal(loc: -0.2017146348953247, scale: 1.004259467124939), 'sample98': Normal(loc
3766495883464813, scale: 0.9975871443748474), 'sample23': Normal(loc: -0.3070654273033142, scale: 0.98941212
235), 'sample96': Normal(loc: 0.16982457041740417, scale: 0.9840357899665833), 'sample83': Normal(loc: 0.341
820144653, scale: 0.9283028841018677), 'sample28': Normal(loc: -0.28709498047828674, scale: 0.87845247983932
sample117': Normal(loc: 0.12659983336925507, scale: 1.0993764400482178), 'sample27': Normal(loc: -0.37188762
634, scale: 1.1278619766235352), 'sample88': Normal(loc: -0.3917408585548401, scale: 1.0515893697738647), 's
116': Normal(loc: 0.19242410361766815, scale: 1.1206859350204468), 'sample5': Normal(loc: 0.3104699552059173
ale: 0.896874725818634), 'sample52': Normal(loc: -0.09707590937614441, scale: 1.0181375741958618), 'sample64
rmal(loc: 0.12425005435943604, scale: 1.166978120803833), 'sample46': Normal(loc: -0.05304451286792755, scal
9874014258384705), 'sample129': Normal(loc: 0.02255052514374256, scale: 1.0709550380706787), 'sample57': Nor
oc: 0.07677019387483597, scale: 0.9037127494812012), 'sample53': Normal(loc: 0.3087914288043976, scale: 1.02
441467285), 'sample12': Normal(loc: 0.1744071990251541, scale: 0.917870819568634), 'sample126': Normal(loc:
423068091273308, scale: 1.1834315061569214), 'sample93': Normal(loc: -0.03997012972831726, scale: 0.96115642
6), 'sample24': Normal(loc: 0.13767124712467194, scale: 1.0464117527008057), 'sample65': Normal(loc: 0.08828
3369217, scale: 0.9979518055915833), 'sample138': Normal(loc: -0.009301628917455673, scale: 0.93537604808807
'sample29': Normal(loc: 0.22290989756584167, scale: 1.03630530834198), 'sample34': Normal(loc: -0.1827595531
02, scale: 0.9403443932533264), 'sample133': Normal(loc: 0.14275957643985748, scale: 1.0136116743087769), 's
119': Normal(loc: -0.4713114798069, scale: 1.0176291465759277), 'sample86': Normal(loc: 0.1443261206150055,
: 1.0197396278381348), 'sample136': Normal(loc: -0.03695037588477135, scale: 1.129204273223877), 'sample58':
al(loc: 0.21542195975780487, scale: 0.9976047277450562), 'sample127': Normal(loc: 0.22007790207862854, scale
459568858146667), 'sample66': Normal(loc: -0.48271283507347107, scale: 1.0355836153030396), 'sample95': Norm
c: -0.29609158635139465, scale: 1.010400652885437), 'sample56': Normal(loc: 0.053470637649297714, scale: 0.9
7288894653), 'sample92': Normal(loc: 0.16050845384597778, scale: 1.168870449066162), 'sample48': Normal(loc:
424364298582077, scale: 1.1600818634033203), 'sample111': Normal(loc: -0.27430272102355957, scale: 1.1669081
8667), 'sample10': Normal(loc: 0.4499799907207489, scale: 0.9570946097373962)}
the max gradient is  35.292686
```

6 Program5

7 Program3

7.1 results