CPSC 532W Homework 4

Naomi Graham

March 9, 2021

All the code can be found on: `https://github.com/n6graham/cpsc532_hw4`.

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: 8.311932563781738, scale:
1.0169004201889038)}

```

But the posterior mean and variance returned by importance sampling were:
tensor(1.7418) Variance: tensor(2.1257)

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.

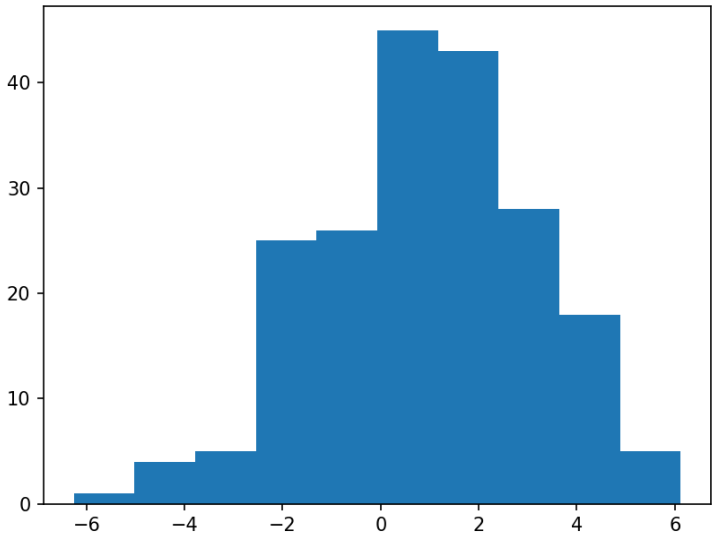
```
    sigma = {'Q':{}, 'logW':0, 'G':{} }
    weighted_samples = []

    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)}
```

But the sampled values look wrong:



3 Program2

4 Program3

5 Program4

6 Program5

7 Program3

7.1 results