Experiment record:

Baseline:

(fashion in paper)

EWC: 42.24 ModEWC: 62.47

GEM: 56.86 ModGEM: 58.47

(the experiment of AGEM that I did, didn't use the optimal hyperparameter)

AGEM: old version (but add the vector length) 0.5855

New project2cone2 function version: 0.6100

Old project2cone2 function but with torch.mean strategy: 0.5804

ModAGEM experiments:

(1) ***Dataset:*** fashion dataset

Note: h_=np.zero(t) where t=pp.shape[0] but we don't use margin here (margin=0)

the key point: standardization (mean=1)

the reason for standardization and relatedness as weight: we want to get the average gradient of previous tasks (stored in pp here) in AGEM and we also have relatedness between tasks, so I just used the relatedness as the weight for gradients. Intuitively, the neuron which has higher relatedness should contribute more to the final weighted average gradient. Besides, we don't use relatedness for the constraint of QP function. Instead, we just use the same way as AGEM (np.zeros). In my opinion, the main structure of QP solver function (project2cone2) in ModAGEM should remain the same as AGEM. So I think it's better to use relatedness along with gradients of previous tasks instead of putting relatedness in the constraint condition of QP solver. I didn't notice the relatedness is around 10 before, so standardization should be used here to make sense. Moving the mean of standardization to 1, weighting the previous gradients with standardized relatedness and finally get average is the main idea here.

result: 0.6231

```
#h = (h-np.mean(h))/np.var(h)   #Standardization of h
if np.var(h)==0:
    h = h-np.mean(h)+1
    pp = np.dot(np.expand_dims(h,axis=0),pp)/pp.shape[0]
else:
    h = (h-np.mean(h))/np.var(h)+1
    pp = np.dot(np.expand_dims(h,axis=0),pp)/pp.shape[0]
```

```
h_ = np.zeros(t)
G = np.eye(t)
G = np.eye(t) + np.eye(t)*0.00001
try:
    v = quadprog.solve_qp(P, q, G, h_)[0]
```

(2) ***Dataset:*** fashion dataset

Note: h_=np.zero(t) where t=pp.shape[0] but we don't use margin here (margin=0)

Main change from (1): keep the mean of standardization to be 0 so there's no need to average over pp.shape[0]

result: 0.5564

```
if np.var(h)==0:
    h = h-np.mean(h)+1
    pp = np.dot(np.expand_dims(h,axis=0),pp)/pp.shape[0]
else:
    h = (h-np.mean(h))/np.var(h)
    pp = np.dot(np.expand_dims(h,axis=0),pp)

#print("pp shape:{}".format(pp))
#pp = np.dot(np.expand_dims(h,axis=0),pp)/pp.shape[0]

P = np.dot(pp, pp.transpose())
P = 0.5 * (P + P.transpose()) + np.eye(P.shape[0])*(1e-3)
q =  np.dot(gradient.cpu().contiguous().view(-1).double().numpy(), pp.transpose())
#t = len(h)
t=pp.shape[0]
h_ = np.zeros(t)
G = np.eye(t)
G = np.eye(t) + np.eye(t)*0.00001
```

(3) ***Dataset:*** fashion datset
Main change: directly average over h and average over pp and use h for QP (this is the most direct change from AGEM to ModAGEM)

```
pp = np.expand_dims(np.mean(pp,axis=0),0)
h = np.expand_dims(np.mean(h),0)  #directly average over h
```

```
v = quadprog.solve_qp(P, q, G, h)[0]
```

result: 0.6557 (as you can see the accuracy for this case is highest among all the experiments, but as it directly averages over relatedness, it is difficult to interpret why we do the average over relatedness)

(4) ***Dataset:*** mnist_rotations_reduced.pt
Main change: directly average over h and average over pp and use h for QP (same as (3))
result: 0.7429

(5) ***Dataset:*** fashion dataset
Main operations: h*pp, h_=np.zero(t) margin=0
Main difference from (1): no standardization and no average
result: 0.6288

(6) ***Dataset:*** fashion datset
Main operations: h*pp/pp.shape[0], h_=np.zero(t) margin=0
Main difference from (1): no standardization
result: 0.6302

Conclusion:

Even though the result of the first experiment is not the best, it makes the most sense. Considering the result of the most experiments are 0.62 or 0.63, I think experiment 1 should be the implementation. I also read some literatures to find other ways to make use of relatedness. But after reading them, I still think weight the previous gradients with relatedness is the best way. The idea is very similar with that in paper TAG: Task-based Accumulated Gradients for

Lifelong Learning. In that paper, the author used a weight matrix including the relatedness times previous gradients to update the learning rate of optimizer.