There are three ways of AGEM:

1. Old version:

accumulate losses of previous tasks and get the gradient based on the accumulated loss and store it in grad[0]

A screenshot of a computer

Description automatically generated with medium confidence

Store the gradient of the current task in grad[1]

Text

Description automatically generated

Still use the same project2cone2 function as GEM

Text

Description automatically generated

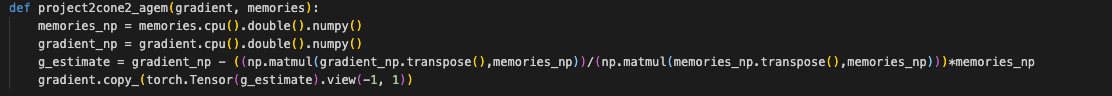
Result:

Text

Description automatically generated

1. New Version:

Use new project2cone2 function according to the formula in AGEM paper



The way to get loss and gradient of previous tasks remains the same as GEM

Text

Description automatically generated

Use the average value of gradients of previous task instead, so notice that the argument memories in new project2cone2 function has the shape [p,1]

Text

Description automatically generated

Result:

Text

Description automatically generated

1. Another way:

We still use the project2cone2 function and the way to get the loss and gradient of previous tasks. The only change is we directly put the mean value of previous gradients into project2cone2 function to use QP method to solve.

Text

Description automatically generated

Result:

Text

Description automatically generated

All the experiments above use the same command call:

python main.py --model Agem --n\_tasks 10 --lr 0.01 --n\_memories 10 --memory\_strength 10 --n\_layers 2 --n\_hiddens 100 --data\_path data/ --save\_path results/ --batch\_size 1 --log\_every 100 --samples\_per\_task 1000 --cuda no --seed 3 --beta 0.03 --gamma 1.0 --memories 200 --replay\_batch\_size 10 --batches\_per\_example 10 --forgetting\_mode False --forgetting\_task\_ids 0,5 --forgetting\_resee\_size 100 --sign\_attacked -1.0 --num\_groups 20 --cov\_recompute\_every 20 --create\_random\_groups False --divergence von\_Neumann --if\_output\_cov False --cov\_first\_task\_buffer 100 --data\_file fashion\_mnist\_permutations\_reduced.pt --ewc\_reverse False --create\_group\_per\_unit False

1. The difficulty of revising modularized AGEM in the three cases above:

For all the three ways mention above, the key revision is what kind of memories we use. But as you can see from the figure below, the argument “memories” actually is not involved in the calculation of the function. Instead, pp acts like memories while h stores information about relatedness.

Text

Description automatically generated

For the old version: because there are only two indexes involved, it can’t be aligned to gradient groups or relatedness groups.

For the new version: because we use a new project2cone2 function, it’s really difficult to add relatedness information (h) and memories (pp) into the new function.

For the another way: because we have to get average of the previous gradients (memories or pp here), the size of pp now is [41, 89610], since the first index corresponds to the number of tasks, we have to average over tasks then pp’s shape will become [1,89610], in order to keep the consistence of shape, we also have to transform h (shape: [41,1]) to [1,1]. I am not sure if this idea is correct because average of h means average of relatedness. Also, the code didn’t work right. The error is called Killed:9.



1. Conclusion:

Since we should modularize AGEM based the correct AGEM code, I think we should first determine which way is correct. Then we can think about the subsequent steps.