ModAGEM Explanation & Experiments

1. Main idea

First, let's see the QP problem in GEM.

$$\min \ g^T G^T v + \frac{1}{2} v^T G G^T v \ \ s.t. \ \ v \geq 0 \qquad (1)$$

We aim to use QP solver function to get the optimal v, then

$$\tilde{g} = G^T v^* + g \qquad (2)$$

Where $\tilde{g}$ is the projected gradient which should be used for the current task, g is the gradient which has been achieved by the training process and G contains the gradients of previous tasks.

Then let's see the standard QP function.

$$\min \ -d^T v + \frac{1}{2} v^T D v \ \ s.t. \ \ A^T v \geq v_0 \qquad (3)$$

For the Python package quadprog.solve_qp(), there are four arguments: D, d, A, m as shown in equation (3).

Comparing equation (1) and (3), $D = GG^T, d = -Gg, A = E, m = 0$.

So for GEM, we just need to update the gradient with the formula above after using QP solver function. The most difference between ModGEM and GEM is that we created groups of 20 (hyperparameter: number of groups) for each layer. Specifically, we store the gradients of previous tasks and relatedness between tasks additionally. For example, when the current task is task 2 (second task) and we only assume there are two layers in the model, the shape of relatedness is (41,) where the relatedness between the first layer of task 1 and task 2 offers 20 values while the relatedness between the second layer of task 1 and task 2 offers the other 20 values.

Let's go back to equation (3). For ModGEM, we chose $D = GG^T, d = -Gg, A = E, m = relatedness$. Note that the gradients of previous tasks (denoted by G) and relatedness (denoted by m here) is stored by group.

So far I have explained the difference between GEM and ModGEM. Now let's see AGEM and ModAGEM.

The difference between AGEM and GEM is that AGEM uses average gradient instead of individual gradients to update the gradient of the current task. But we had three versions of code. For the first one (what Ammar did), we still use QP solver function to update gradient. During the training, we accumulate losses, use it to get average gradient and store it in index 0 while the gradient of the current task is stored in index 1. For the second version (what I did), we still keep the training period the same and have the indexes of the number of tasks. Then we average over the gradients of pervious tasks and use new formula equation (4) to update gradient. For the third version, we just replace the update function from equation (4) to the QP solver equation.

$$\tilde{g} = g - \frac{g^T g_{average}}{g_{average}^T g_{average}} g_{average} \qquad (4)$$

For ModAGEM, if we directly use the idea of AGEM, we should average over gradients of previous tasks (denoted by G), in order to keep the shape consistent, the relatedness (denoted by m) should also be averaged. In this case, the meaning of the relatedness seems vague. Similar with the idea of paper TAG: Task-based Accumulated Gradients for Lifelong Learning, I weight G with relatedness and let m=0 again like GEM and AGEM. Intuitively, the neuron which has higher relatedness should contribute more to the final weighted average gradient.

Hence, for ModAGEM, the argument in equation (3) should be $D = hGG^T h^T$, $d = -hGg$, $A = E, m = 0$. Note that h is relatedness. In order to make sure introducing of weight for G will not lead to a huge fluctuation, relatedness (denoted by h) should be standardized and the mean should be moved from 0 to 1.

To get average gradient of previous tasks just like AGEM, we finally divide hG by the total number of groups.

2. Experiment

Baseline:

(the performance of fashion dataset in paper with optimal hyperparameters)

EWC: 42.24 ModEWC: 62.47

GEM: 56.86 ModGEM: 58.47

AGEM: first version: 58.55

second version: 61.00

third version: 58.04

(note that I didn't use optimal hyperparameters for AGEM and you can find the description of three versions in Main idea above)

ModAGEM: (not optimal hyperparameters)

(1) The implementation of ModAGEM idea in Main idea above
Result: 62.31
(2) If we don't move the mean of standardization from 0 to 1 and don't average over total number of groups (because if the mean of standardization is 0, we actually don't need average)
Result: 55.64
(3) If we directly average over relatedness, use it as m and don't weight gradients of previous tasks with relatedness just like what ModGEM did
Result: 65.57 (although highest, make little sense)
(4) If we don't use standardization and average over total number of groups
Result: 62.88
(5) If we don't use standardization only
Result: 63.02

Example call:

```
python main.py --model Agem_GMod2Cov --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
```