Appendix of Dealing with Changing Training Bias in Replay for Online Continual Learning

Anonymous CVPR submission

Paper ID 10165

1. Appendix 1. Sub-figures (D), (E) and (F) for Figure 1 of the Main Paper

The sub-figures are shown in Figure 1 here.

2. Appendix 2: Figure of the average cross-entropy loss of the new data batch and the replay data batch

We plot the average loss in Figure 2 and observe that the average cross-entropy loss of the new data batch is consistently higher than the loss of the replay data batch.

3. Appendix 3. Sub-figures (D), (E), and (F) Using the SGD Optimizer for Figure 2 of the Main Paper

The sub-figures are shown in Figure 3 here.

4. Appendix 4: Proof for the upper bound

Assume that the system has seen $n-1$ previous classes $(c_1, ..., c_{n-1})$ and there are $m - n$ new classes $(c_n, ..., c_m)$ in the current new data batch. (1) It’s easy to see that $L_{\text{decom}}(x_{c_n}) = L_{\text{ce}}(x_{c_n})$ if $m - n = 1$ because both losses degenerate to just establishing the boundary between the new class and old classes and their forms are the same. (2) It is also easy to see that $L_{\text{decom}}(x_{c_n}) = L_{\text{ce}}(x_{c_n})$ when $n - 1 = 0$ because both losses degenerate to just establishing the decision boundaries between the new classes within the new data batch and their formulas become the same. (3) So let us consider the case that $m - n > 1$ and $n - 1 > 0$. For a sample $x_{c_n}$ of class $c_n$ from $X_{\text{new}}$, its cross-entropy loss is:

$$L_{\text{ce}}(x_{c_n}) = -\log \left( \frac{e^{o_{c_n}}}{\sum_{s=1}^{m} e^{o_{c_s}}} \right)$$  

(1)

where $o_{c_s}$ is the logit value of $x_{c_n}$ for class $c_s$. Then we have:

$$L_{\text{ce}}(x_{c_n}) = -\log \left( \frac{e^{o_{c_n}}}{\sum_{s=1}^{n-1} e^{o_{c_s}}} \right)$$

(2)

$$L_{\text{ce}}(x_{c_n}) = -\log \left( \frac{e^{o_{c_n}}}{\sum_{s=n}^{m} e^{o_{c_s}}} \right)$$

(3)

Combining Eq. 2 and Eq. 3, we get:

$$L_{\text{ce}}(x_{c_n}) = -\frac{1}{2} \log \left( \frac{e^{o_{c_n}}}{\sum_{s=n}^{m} e^{o_{c_s}}} \right) - \frac{1}{2} \log \left( \frac{\sum_{s=1}^{m} e^{o_{c_s}}}{\sum_{s=n}^{m} e^{o_{c_s}}} \right)$$

(4)

For the loss $L_{\text{decom}}(x_{c_n})$, we have:

$$L_{\text{decom}}(x_{c_n}) = -\frac{1}{2} \log \left( \frac{e^{o_{c_n}}}{\sum_{s=1}^{m} e^{o_{c_s}}} \right) - \frac{1}{2} \log \left( \frac{\sum_{s=1}^{m} e^{o_{c_s}}}{\sum_{s=n}^{m} e^{o_{c_s}}} \right)$$

(5)

Considering the second term in Eq. 4 and the last term in Eq. 5, we have:

$$\frac{e^{o_{c_n}}}{\sum_{s=1}^{m} e^{o_{c_s}}} = 1 - \frac{\sum_{s=n}^{m} e^{o_{c_s}}}{\sum_{s=1}^{m} e^{o_{c_s}}}$$

(6)

As $\sum_{s=n+1}^{m} e^{o_{c_s}} \geq 0$, so we have:

$$\frac{e^{o_{c_n}}}{\sum_{s=1}^{m} e^{o_{c_s}}} \leq \frac{\sum_{s=n}^{m} e^{o_{c_s}}}{\sum_{s=1}^{m} e^{o_{c_s}}}$$

(7)
The memory buffer size is 1000. We choose four different tasks and plot their PN rates as subsequent tasks are learned. In (F), we plot the test accuracy in the CIFAR100 experiment using the SGD optimizer.

5. Appendix 5: Justification for the GSA-CE Loss

Assuming \( x_i \) is a data sample of class \( c_i \), the gradient on each logit of \( o(x_i; \theta, \phi) \) is given by:

\[
\frac{\partial \mathcal{L}_{ce}(o(x_i; \theta, \phi))}{\partial o_{c_i}} = \frac{1}{e^{A-PN(t, t, y_k) + 1}} \cdot H(o_{c_i}) \tag{10}
\]

where \( H(o_{c_i}) = \frac{e^{o_{c_i}}}{\sum_{x=1}^{[C_{obs}]} e^{o_{c_x}} + \frac{e^{o_{c_i}}}{PN(t, c_i)}} - 1 \) and for \( j \neq i \)

\[
\frac{\partial \mathcal{L}_{ce}(o(x_i; \theta, \phi))}{\partial o_{c_j}} = \frac{1}{e^{A-PN(t, t, y_k) + 1}} \cdot H(o_{c_j}) \tag{11}
\]

where \( H(o_{c_j}) = \frac{e^{o_{c_j}}}{\sum_{x=1}^{[C_{obs}]} e^{o_{c_x}} + \frac{e^{o_{c_j}}}{PN(t, c_j)}} \). For a class \( c_i \), the smaller its accumulated gradient rate \((A-PN(t', t, c_i))\) is, the bigger its sample weight \( w_i \) is. So the model can automatically adjust the loss weight \( w_i \) to emphasize the class that has a smaller accumulated gradient rate because the smaller accumulated gradient means that the model underrates this class. However, since the accumulated gradient rate \( A-PN \) is calculated from all seen training data, it is not sensitive to short-term tendency. To capture short-term variations within a task, we introduce \( \frac{1}{PN(t, c_i)} \) to the logit of \( c_i \). From Eq. 10 and Eq. 11 above, we know that if the positive gradient of \( c_i \) within a task surpasses the negative gradient of \( c_i \) within the task (\( \frac{1}{PN(t, c_i)} < 1 \)), the negative gradient will be adjusted by increasing \( \frac{1}{PN(t, c_i)} \) until \( \frac{1}{PN(t, c_i)} = 1 \) and the positive gradient for the logits of other classes (not \( c_i \)) will be increased to help the model distinguish samples of other classes, and vice versa.

6. Appendix 6: Official Code of Baselines, Hyperparameters

Following [12], we set each data batch \( (X^{new}, N^{new}) \) size \( N^{new} \) to 10 for all systems. We use \( X^{new} \) as the input of group
2. For the input of group 1 \( (X^{\text{mix}}) \), we set it to 64 in GSA. Specifically, we first sample \( \max(int(64 \cdot \frac{1}{\sum_{t=1}^{L} N^t}), 1) \) data points from \( X^{\text{new}} \) and sample \( N^{\text{buf}} = 64 - N^{\text{new}} \) data points from the memory buffer, and then we combine them to produce \( X^{\text{mix}} \). For fair comparison, for baselines, we also set their memory buffer batch \( (X^{\text{buf}}) \) size to 64 (it does not change with tasks). We use the Apex (A PyTorch Extension) \url{https://nvidia.github.io/apex/} to accelerate training for all methods. “opt_level” is “O1”. We run all experiments on a Tesla V100 32G. The code links for baselines are

- The code for ER and MIR: \url{https://github.com/Optimass/Maximally_Interfered_Sampling}
- The code for ASER and SCR: \url{https://github.com/RaptorMai/online-continual-learning}
- The code for GDumb: \url{https://github.com/drimpossible/GDumb}
- The code for BiC: \url{https://github.com/pclucas14/AML}
- The code for SSIL: \url{https://github.com/sairin1202/BiC}
- The code for CCIL: \url{https://github.com/rahafaljundi/Gradient-based-Sample-Selection}
- The code for AGEM: \url{https://github.com/hshustc/CVPR19_Incremental_Learning}
- The code for LUCIR: \url{https://github.com/Mirandaga/Incremental_Continual_Learning}
- The code for BiC: \url{https://github.com/sairin1202/BiC}
- The code for SSIL: \url{https://github.com/sairin1202/SSIL-Official}

### 7. Appendix 7: Forgetting Rate and Training time

The average forgetting rate is computed as follows [7]: after training the model from task 1 to task \( j \), we denote \( acc_{c,i} \) as the accuracy of the trained model evaluated on the held-out test set of task \( i \leq j \). The average forgetting rate \( FR_t \) at task \( t \) is:

\[
FR_t = \frac{\sum_{i=1}^{t-1} f_i^t}{t-1}, \quad \text{where} \quad f_i^t = \max_{t \in \{1, 2, ..., t-1\}} (acc_{c,i} - acc_{c,i})
\]

Training times of our system GSA and the baselines are given in Figure 4. Our method is slower than some baseline systems due to the extra computation needed to ensure gradient balance. However, with the ever-increase in computing power of new hardware, this extra computation should not be a major issue. Some code optimization should also improve our algorithm greatly.

### Table 1. Accuracy of the long-tailed CIFAR100 (20 tasks) experiment with different memory buffer sizes \( M \). All values are averages of 15 runs.

<table>
<thead>
<tr>
<th>Method</th>
<th>Long-tailed CIFAR100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( M=0.1k )</td>
</tr>
<tr>
<td>AGEM [5]</td>
<td>6.5 ± 0.2</td>
</tr>
<tr>
<td>ER [6]</td>
<td>7.0 ± 0.1</td>
</tr>
<tr>
<td>MR [7]</td>
<td>10.1 ± 0.1</td>
</tr>
<tr>
<td>GSA [8]</td>
<td>7.4 ± 0.5</td>
</tr>
<tr>
<td>ASER [9]</td>
<td>9.9 ± 0.1</td>
</tr>
<tr>
<td>LUCIR [10]</td>
<td>8.5 ± 0.3</td>
</tr>
<tr>
<td>SCR [11]</td>
<td>9.2 ± 0.3</td>
</tr>
<tr>
<td>GSA [12]</td>
<td>15.8 ± 0.3</td>
</tr>
<tr>
<td>LUCIR [13]</td>
<td>13.1 ± 0.2</td>
</tr>
<tr>
<td>CCIL [14]</td>
<td>21.9 ± 0.2</td>
</tr>
<tr>
<td>BiC [15]</td>
<td>23.2 ± 0.2</td>
</tr>
<tr>
<td>GSA [16]</td>
<td>29.1 ± 0.6</td>
</tr>
</tbody>
</table>

Figure 4. Training times of our method GSA and baselines on CIFAR10 in minutes.
8. Appendix 8: Long-tail Online Continual Learning Experiments Setup

This setting has 20 tasks created from the CIFAR100 dataset and each task consists of 5 randomly chosen classes from the 100 classes of CIFAR100. The number of training samples $n_c$ in each class $c$ is

$$n_c = n_{c,orig} \times u^{\text{index}(c)}$$

where $n_{c,orig}$ is the original number of training samples for class $c$ in the CIFAR100 dataset and $\text{index}(c)$ is the class index of class $c$ (0-indexed) and $u \in (0, 1)$. We set $u$ as 0.98. The backbone and the optimizer are the same as those in the main experiments. We list the final average test performance in Table 1. From the table, we observe that our method outperforms the baselines over three different memory sizes.

References


