持续学习项目报告

Gradient Episodic Memory (LCL-8)

211250062 吴陈添
211250063 喻重阳
211250117 周博龄
211250127 梁安然
211250165 刘尧力

2023 年秋 机器学习

南京大学

2024 年 1 月 2 日

目录

1. 项目简介

我们选择的论文 [1] 提出了 Gradient Episodic Memory (GEM) 模型, 用于解决连续学习中的遗忘问题. 该模型在每次学习新任务时, 会将新任务的梯度与之前任务的梯度进行比较, 并将新任务的梯度投影到之前任务的梯度上, 从而保证之前任务的梯度不会被遗忘.

为了完成这个项目, 我们开展了如下的几个步骤:

1. 阅读论文, 理解论文的内容
2. 运行代码
3. 阅读论文的代码, 结合代码理解论文的实现
4. 阅读 LibContinual 的代码
5. 将 GEM 模型结合到 LibContinual 中
6. 调试, 改进

我们的代码在 NJU GitLab 仓库 中给出. 本次报告中出现的所有符号在附录A中给出.

2. 论文和 GEM 模型分析

2.1. 监督学习的背景

一般来说, 监督学习的第一步是找到训练集 $D_{\text{tr}} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$, 其中, $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ 是由 $\boldsymbol{x}_i \in \mathcal{X}$ 和 $\boldsymbol{y}_i \in \mathcal{Y}$ 构成的样本, $\mathcal{X}$ 是特征向量集, $\mathcal{Y}$ 是目标向量集, n 是样本数量. 绝大部分监督学习都假设, 训练集 D_{tr} 是由一个固定的概率分布 P 生成的, 即 $(x_i, y_i) \sim P$, 并且假设样本之间是独立同分布 (independently distributed, iid) 的.

因此, 监督学习的目标就是, 通过训练集 D_{tr} 来得到这样的一个函数, 或者叫模型 $f : \mathcal{X} \to \mathcal{Y}$, 使得模型 f 能够对新的样本 $\boldsymbol{x} \in \mathcal{X}$ 给出预测值 $\boldsymbol{y} \in \mathcal{Y}$. 可以认为, f 和 P 是相互对应的, 也就是说 f 是 P 的一个估计.

为了得到这样的 f, 监督学习通常采用经验风险最小化 (Empirical Risk Minimization, ERM) 的方法. 经验风险被定义为

$$\hat{R}_{\text{emp}}(f) = \frac{1}{\|D_{\text{tr}}\|} \sum_{(\boldsymbol{x}_i, \boldsymbol{y}_i) \in D_{\text{tr}}} \ell(f(\boldsymbol{x}_i), \boldsymbol{y}_i),$$

其中, $\ell : f(\mathcal{X}) \times \mathcal{Y} \to [\,0, +\infty\,)$ 是损失函数, 能够衡量模型 f 在给定的特征向量 $\boldsymbol{x}$ 上的预测值 $f(\boldsymbol{x})$ 与真实值 $\boldsymbol{y}$ 的差距. 一般来说, 损失函数 l 是非负的, 并且当且仅当 $f(\boldsymbol{x}) = \boldsymbol{y}$ 时取到最小值.

然而, ERM 有一个重要的假设是样本独立同分布, 但是在现实中这个假设往往是不成立的. 正如论文 [2] 中指出, 直接应用 ERM 会导致灾难性遗忘 (catastrophic forgetting), 即在学习新任务时, 会大量**忘记**之前学习的任务.

2.2. 持续学习

在这样的背景下, 持续学习 (Continual Learning, CL) 的概念被提出. 持续学习的目标是, 在学习新任务时, 尽可能地保留之前学习的任务. 持续学习关注这样的数据连续体

$$(\boldsymbol{x}_i, t_i, \boldsymbol{y}_i) \quad i \in \mathbb{N}^+,$$

其中, $\boldsymbol{x_i} \in \mathcal{X}_{t_i}$ 是特征向量, $t_i \in \mathcal{T}$ 是任务标签, $\boldsymbol{y_i} \in \mathcal{Y}$ 是目标向量, $\mathcal{T}$ 是任务标签集. 我们一般认为上面说的数据连续体是局部独立同分布的, 也就是

$$(\boldsymbol{x}_i, \boldsymbol{y}_i) \overset{\text{iid}}{\sim} P_{t_i}(X, Y)$$

基于这样的假设, 我们的目标是找到一个函数 $f : \mathcal{X} \times \mathcal{T} \to \mathcal{Y}$, 使得对于任意的 $t \in \mathcal{T}$, f 都能够在任务 t 上给出一个好的预测.

2.3. 本文目标

训练协议和评估指标大多数关于学习一系列任务的文献 [3, 4, 5, 6] 都使用以下训练协议和评估指标. 他们假设

1. 任务数量很少;
2. 每个任务的示例数量很大;
3. 学习者对每项任务的示例进行多次遍历;
4. 报告的唯一指标是所有任务的平均性能.

相比之下, 本文则采用" 更像人类" 的训练协议和评估指标, 即:

1. 任务数量很大;
2. 每个任务的示例数量很少;
3. 学习者仅观察一次有关每项任务的示例;
4. 同时需要衡量转移和遗忘的指标.

更具体地说, 假设总共有 T 个任务, 构建矩阵 $R \in \mathbb{R}^{T \times T}$, 其中 $R_{i,j} \in \{0, 1\}$ 表示在学习完第 $t_i \le T$ 个任务的最后一个样本之后, 预测第 $t_j \le T$ 个任务的正确性. $\bar{b}$ 表示每个任务测试准确率的随机初始值. 我们会评估以下指标:

平均准确率 (average accuracy, ACC), 即所有任务的平均准确率.

$$\text{ACC} := \frac{1}{T} \sum_{i=1}^{T} R_{T,i}$$

向后迁移 (backward transfer, BWT), 即学习任务 t 对前一个任务 $k \prec t$ 的性能的影响. 一方面, 当学习某些任务 t 提高了某些先前任务 k 的性能时, 存在正向后迁移. 另一方面, 当学习某些任务 t 会降低某些先前任务 k 的性能时, 就会存在负向后迁移. 大的负向后转移也称为灾难性遗忘 (catastrophic forgetting).

$$\text{BWT} := \frac{1}{T-1} \sum_{i=1}^{T-1} R_{T,i} - R_{i,i}$$

前向迁移 (forward transfer, FWT), 即学习任务 t 对未来任务 $k \succ t$ 的性能的影响. 特别是, 当模型能够执行" 零样本" 学习时, 可能会通过利用任务描述符中可用的结构来实现正向前向迁移.

$$\text{FWT} := \frac{1}{T-1} \sum_{i=2}^{T} R_{i-1,i} - \bar{b}_i$$

2.4. GEM 模型

在本节中, 本文提出了一种新的持续学习模型梯度情景记忆 (Gradient Episodic Memory, GEM). GEM 的主要特征是情景记忆 $\mathcal{M}_t$, 它存储任务 t 中观察到的样本的子集. 为了简单起见, 作者假设整数任务描述符, 并使用它们来索引情景内存. 当使用整数任务描述符时, 不能期望显着的正向传递 (零样本学习). 相反, 本文专注于通过有效使用情景记忆来最大限度地减少负向后转移 (灾难性遗忘).

在实践中, 学习者的总预算是 M 个记忆位置. 如果总任务数 T 已知, 我们可以为每个任务分配 $m = M/T$ 的记忆容量. 相反, 如果任务总数 T 未知, 我们可以在观察新任务时逐渐减小 m 的值 [6]. 为了简单起见, 作者假设内存中填充了每个任务的最后 m 个示例, 尽管可以采用更好的内存更新策略 (例如为每个任务构建一个核心集). 接下来, 考虑由 $\theta \in \mathbb{R}^p$ 参数化的预测变量 f_θ, 并将第 k 个任务的**记忆损失**定义为:

$$\ell\left(f_\theta, \mathcal{M}_k\right) = \frac{1}{\|\mathcal{M}_k\|} \sum_{\left(\boldsymbol{x}_i, k, \boldsymbol{y}_i\right) \in \mathcal{M}_k} \ell\left(f_\theta\left(\boldsymbol{x}_i, k\right), \boldsymbol{y}_i\right)$$

GEM 的训练个评估伪代码如算法 1所示. 与此前的持续学习方法相比, GEM 的具体策略如下:

调整优化策略: 我们使用记忆损失来调整经验风险, 使其包括对过去任务的记忆损失. 因此, 当观察到一个三元组 $(\boldsymbol{x}, t, \boldsymbol{y})$ 时, GEM 模型希望能够实现**学习最新的任务之后, 对于之前学习过的每一个任务, 记忆损失都不能再增加, 并在此基础上尽可能减少现有的记忆损失**. 形式化地, GEM 的目标是在满足

$$\forall k < t : \ell(f_\theta, \mathcal{M}_k) < \ell(f_\theta^{t-1}, \mathcal{M}_k)$$

的条件下, 最小化

$$\ell\left(f_\theta(\boldsymbol{x}, t), \boldsymbol{y}\right)$$

其中, f_θ^{t-1} 是在学习完任务 $t-1$ 后的模型.

记忆约束: 为了满足上述条件, 我们需要在每次更新参数 θ 时, 通过**记忆约束**来确保记忆损失不会增加. 具体来说, 我们希望在每次更新参数 θ 时, 都能够找到一个梯度 g, 使得:

$$\forall k < t : \langle g, g_k \rangle := \left\langle \frac{\partial \ell\left(f_\theta(\boldsymbol{x}, t), \boldsymbol{y}\right)}{\partial \theta}, \frac{\partial \ell\left(f_\theta, \mathcal{M}_k\right)}{\partial \theta} \right\rangle \geq 0$$

直观上就是说, 在更新参数 θ 时, 我们希望学习最新的任务之后的损失函数和此前的损失函数具有相同的变化趋势. 不难发现这是一种更强的限制.

梯度投影: 不难发现, 上面的约束并不总是能满足的, 尤其是当相邻的两组任务之间的冲突很大时. 如果不能找到这样的梯度 g, 我们就将 g 投影到一个和它最接近的, 并且满足记忆约束要求的梯度 $\tilde{g}$, 使得在

$$\forall k < t : \langle \tilde{g}, g_k \rangle \geq 0$$

的前提下, 调整 $\tilde{g}$ 的值, 使得 $\|g - \tilde{g}\|$ 的值最小. 接下来使用二次规划来求解这个问题.

Algorithm 1 在有序的数据连续体上训练 GEM 模型

procedure Train(f_θ, Continuum$_{\text{train}}$, Continuum$_{\text{test}}$)
  $\mathcal{M}_t \leftarrow \{\}$ for all $t = 1, \ldots, T$.
  $R \leftarrow 0 \in \mathbb{R}^{T \times T}$.
  for $t = 1, \ldots, T$ **do**:
    for $(\boldsymbol{x}, \boldsymbol{y})$ in Continuum$_{\text{train}}(t)$ **do**
      $\mathcal{M}_t \leftarrow \mathcal{M}_t \cup (\boldsymbol{x}, \boldsymbol{y})$
      $g \leftarrow \nabla_\theta \ell(f_\theta(\boldsymbol{x}, t), \boldsymbol{y})$
      $g_k \leftarrow \nabla_\theta \ell(f_\theta, \mathcal{M}_k)$ for all $k < t$
      $\tilde{g} \leftarrow$ Project$(g, g_1, \ldots, g_{t-1})$
      $\theta \leftarrow \theta - \alpha \tilde{g}$.
    end for
    $R_{t,:} \leftarrow$ Evaluate$(f_\theta,$ Continuum$_{\text{test}})$
  end for
  return f_θ, R
end procedure

procedure Evaluate(f_θ, Continuum)
  $r \leftarrow 0 \in \mathbb{R}^T$
  for $k = 1, \ldots, T$ **do**
    $r_k \leftarrow 0$
    for (x, y) in Continuum(k) **do**
      $r_k \leftarrow r_k +$ accuracy$(f_\theta(\boldsymbol{x}, k), \boldsymbol{y})$
    end for
    $r_k \leftarrow r_k$ / len(Continuum(k))
  end for
  return r
end procedure

3. 结合 GEM 的 LibContinual 模型

3.1. GEM (LCL-8) 项目简介

3.1.1. 项目代码结构

论文项目代码主要结构如下:

```
LCL-8/
├── README.md
├── data/
│   ├── cifar100.py
│   ├── mnist_permutations.py
│   ├── mnist_rotations.py
│   └── raw/
│       └── raw.py
├── main.py
├── metrics/
│   ├── __init__.py
│   └── metrics.py
├── model/
│   ├── __init__.py
│   ├── common.py
│   ├── ewc.py
│   ├── gem.py
│   ├── icarl.py
│   ├── independent.py
│   ├── multimodal.py
│   └── single.py
└── requirements.txt
```

```
└── results
    └── plot_results.py
```

其中 `model/gem.py` 为 GEM 算法的主要实现, `data/`, `metrics/` 分别是数据处理和结果评估部分. 我们接下来分别解析

3.1.2. GEM 算法内部函数解析

主体算法部分 `Net` 类

- `__init__`: 初始化 GEM 算法神经网络的结构和参数, 包括网络结构、损失函数、优化器、内存分配等
- `forward(self, x, t)`: 前向迁移函数. 它确保只预测当前任务内的类别. 通过在输出张量的适当位置填充负无穷大值, 以确保不会选择超出当前任务范围的类别.
- `observe(self, x, t, y)`: 观察函数. 是 GEM 算法的核心, 通过更新内存和学习新示例处理灾难性遗忘问题. 在学习新任务时保存过去任务的梯度, 并在需要时进行投影, 模型可以更好地保留先前学到的知识. 具体分为以下步骤:
  1. 内存更新: 将来自当前任务的一批样本存储在内存中
  2. 先前任务的梯度计算: 遍历已观察到的任务列表, 对内存中存储的过去任务样本执行前向迁移和反向迁移, 计算梯度.
  3. 当前任务的梯度计算: 对当前任务的小批量样本执行前向迁移和反向迁移, 计算梯度.
  4. 梯度约束检查: 如果已经观察了多个任务, 检查当前任务的梯度是否违反了约束. 如果违反, 使用 GEM 投影进行修正.
  5. 应用梯度: 将梯度应用于模型参数, 并进行优化.

剩余辅助函数

- `compute_offsets(task, nc_per_task, is_cifar)`: 算用于数据集的偏移, 确定为给定任务选择哪些输出.
- `store_grad(pp, grads, grad_dims, tid)`: 存储过去任务的参数梯度.
- `overwrite_grad(pp, newgrad, grad_dims)`: 用新的梯度向量覆盖梯度, 以纠正违规.
- `project2cone2(gradient, memories, margin=0.5, eps=1e-3)`: 解决 GEM 论文中描述的二次规划问题, 用于给定提议的梯度和任务梯度内存, 覆盖梯度以进行最终投影更新.

3.2. LibContinual 项目简介

3.2.1. LibContinual 框架架构

通过阅读框架源码, 该 LibContinual 持续学习框架将持续学习的步骤抽象, 分出 5 大模块:

1. *config* 模块: 位于 `config/`, 负责**整个训练过程的参数配置**. 新增算法时, 为每个方法在该路径下新建一个 yaml 文件, 再在 `run_trainer.py` 里填入各自方法对应的 yaml 配置文件路径.

2. *data* 模块: 位于 `core/data`, 负责**数据集的读取**. dataloader.py 负责根据配置信息初始化数据加载器, dataset.py 负责创建和管理连续学习任务的数据集. 可以在 dataloader.py 添加预处理 (transform) 的逻辑.

3. *model* 模块: 位于 `core/model`, 负责**机器学习模型的定义**, 包括 backbone 模型骨架、buffer 模型缓存、replay 模型传播算法三个部分.

4. *trainer* 模块: 位于 `core/trainer.py`, 负责**整个实验的流程控制**.

   (a) 通过 util 类中的 `get_instance ()` 方法, 获取其他模块的对象.

   (b) 目前抽象得不够完善, 新增部分算法时需要修改 `train_loop`函数以实现部分功能.

5. *utils* 模块: `core/utils/` 训练过程中用到的**工具类**. 目前包含 Logger 日志输出类、AverageMeter 指标计算类、字符串格式化方法等工具.

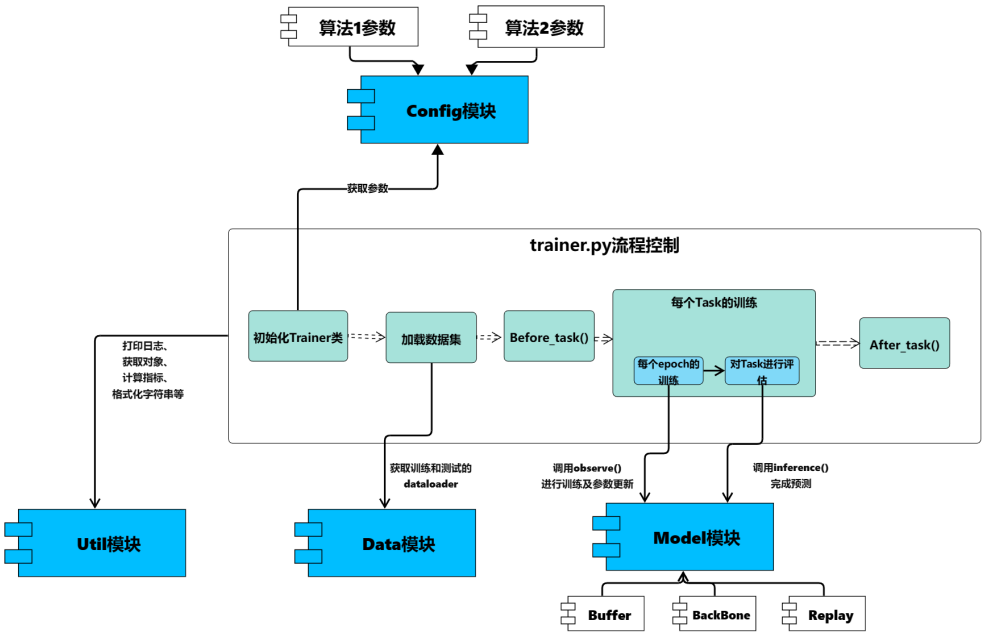


图 1: 模块依赖图

3.2.2. 训练流程解析

入口函数: 框架入口为根目录下的`run_trainer.py`. 修改 `config` 中的配置类即可选择运行测试哪一个持续学习框架.

构建 `class Train`: 进入 `main()` 函数, 首先记录当前时间, 随后从配置类构建 `class Train`. 依次初始化必要信息: 日志记录器 `logger`, 设备 `device` (目前不支持分布式多 gpu

训练)、总任务数 `task_num`, 模型 `model`, 缓冲区 `buffer`, 优化器 `optimizer`, 学习率调度器 `scheduler`, 计量器 `train_meter` 和 `test_meter` 等.

每个 Task 的训练循环: 初始化完成后, 进入主循环 `train_loop()`, **负责每一轮 task 的流程控制**.

1. 在每个 task 开始前, 会执行 `before_task()` 方法, 初始化优化器和学习率调度器, 并调整缓冲区的总类别数. (如果缓冲区的策略是线性缓冲区 (LinearBuffer) , 会将缓冲区的图像和标签添加到数据集中, 并重新创建数据加载器)

2. 进入每个训练周期 epoch 的循环后, 会调用`_train()`函数进行训练, 并打印损失和平均准确率. 如果达到了验证周期 (`val_per_epoch`) 或者是最后一个训练周期, 会调用`_validate()`函数进行测试, 并更新最佳准确率. 随后更新学习率.

   - `_train()`函数: 位于 `core/trainer.py` , **负责每一轮 epochs 的模型训练**. 它接受两个参数: `epoch_idx` 表示当前的训练轮数, dataloader 是一个数据加载器, 用于加载训练数据. 函数内部初始化一个计量器 meter 用于记录训练过程中的指标. 接下来通过一个循环遍历 dataloader 中的每个 batch 数据. 对于每个 batch, 模型会根据输入数据进行前向传播, 计算输出、准确率和损失. 然后使用优化器将损失反向传播并更新模型的参数. 最后, 计量器 meter 更新准确率指标.
   - `_validate()`函数: 位于 `core/trainer.py`, **负责验证当前模型在之前所有任务上的性能**, 防止遗忘. 它接受一个参数 `task_idx`, 表示当前的任务索引. 函数内部首先根据任务索引获取相应的数据加载器 dataloader. 然后初始化一个计量器 meter 用于记录验证过程中的指标. 接下来通过一个循环遍历 dataloader 中的每个 batch 数据. 对于每个 batch, 模型会根据输入数据进行推断, 计算输出和准确率. 然后计量器更新准确率指标. 最后将每个任务的平均准确率和每个任务的准确率列表返回.

3. 在每个任务结束之后, 会执行 `after_task()` 方法, 并根据缓冲区的策略进行缓冲区的更新操作.

循环出口: 结束所有 Task 的训练与测试.

3.2.3. 框架核心部分解析:

即新增持续学习算法的核心函数, 扩展性较强, 包括构成 `model` 的几个模块.

- *replay* 模块: 位于 `core/model/replay`, 负责 **具体持续学习算法的定义**.

  1. `def __init__()`: 用来初始化各自算法需要的对象
  2. `def observe(self, data)`: **训练**过程中, 面对到来的一个 batch 的样本完成训练的损失计算以及参数更新, 返回 `pred`, `acc`, `loss` (预测结果, 准确率, 损失)
  3. `def inference(self, data)`: **推理**过程中, 面对到来的一个 batch 的样本, 完成预测, 返回 `pred`, `acc`
  4. `def before_task() / def after_task()`: 如果算法在每个任务开始前后有额外的操作, 在这两个函数内完成

- *bakcbone* 模块: 位于 `core/model/backbone`, 负责 **backbone 模型文件的定义**.

1. 定义卷积层, 池化层和激活函数, 不负责定义全连接层.
2. 目前只有 `resnet.py` 文件中包含了 `resnet18`, `resnet34`, `resnet50` 等模型骨架, 可以添加其他 `model` 的 `backbone.py`, 以适配不同的持续学习算法

- *buffer* 模块: 位于 `core/model/buffer`, 负责 **训练过程中 buffer 的管理以及更新**.

1. 新增文件以添加 Buffer 类型: 目前实现了 `LinearBuffer` 和 `HerdingBuffer`, 在每个任务开始前会把 buffer 样本与新样本拼接在一起.
2. 更新策略: 在 `update.py`. 目前支持 random 更新和 hearding 更新.

3.3. 将 GEM 模型集成到 LibContinual 中

为了将 GEM 模型加入到 LibContinual 库中, 我们主要进行了如下的修改.

3.3.1. 添加 `gem.py`

`gem.py` 模型主要的控制部分, 用于控制 GEM 算法的训练和推理过程. 值得注意的是, GEM 原本的代码将所有的数据放在一起训练, 并未分成不同的任务. 具体表现在原本的循环写法是这样的:

```
for (i, (x, task_idx, y)) in enumerate(tqdm(dataloader)):
```

而在 LibContinual 中, 我们的训练过程是按照任务进行的, 明显表现在循环写法上是这样的:

```
for task_idx in range(self.task_num):
    for epoch_idx in range(self.init_epochs):
```

总的来说, 原文中 GEM 模型对应的 `class GEM` 具有更多的职责. 在类的外部调用时, 只需要加以简单的刻画, 就可以完成训练和测试. 但是 `LibContinual` 中所有的模型类都被赋予了更有限的职责, 因此我们的主要工作就是对 GEM 的代码解耦合, 将大量的职责分开, 嵌入到给定的库代码的对应部分.

3.3.2. 修改 `trainer.py`

和上述 `gem.py` 对应的, `trainer.py` 中的修改的部分也基本是相应的解耦合. 原本的 GEM 代码更像是结构式编程, 而 `LibContinual` 中的代码更像是面向对象编程. 因此, 我们将原本的代码中的对应功能修改为了类的对应的成员变量和成员函数. 具体可以查看代码.

3.3.3. 其余修改

为了实现上述的两大点修改, 我们还进行了大量的修改, 包括但不限于:

- 添加`ringbuffer.py`作为单独的 buffer 对象.
- 添加`metics.py`用作评价指标.
- 大量修改`dataloader.py`和`dataset.py`中的代码, 以适配 GEM 模型的训练和测试.
- 由于 GEM 模型使用的是二进制 `cifar100` 数据集, 因此我们大量修改了数据的读取, 处理和预处理部分的代码.

3.3.4. 一些尝试

为了从软件工程的角度将 GEM 的代码更好地融入到 LibContinual 中, 我们还尝试了一些其他的方法, 但是由于时间和能力的限制, 并没有成功. 以下是我们进行的尝试.

- `cifar100`有几种不同格式的数据集, LibContinual 使用的是 image 格式, 而 GEM 使用的是 binary 格式的数据集, 我们试着将 binary 数据的处理和 image 数据的处理结合起来, 使得一种方法可以同时处理两种数据, 但是我们并未成功.
- 现在论文中有不少 `if-else` 的写法, 因为 GEM 与 LibContinual 的差距实在太大, 我们难以将所有的 `if-else` 都去掉. 我们尝试利用面向对象函数的重载, 来减少控制流的使用. 由于 Python 中并不支持传统的函数重载, 因此我们充分利用函数参数的默认值, 尽我们的能力减少了 `if-else` 的使用, 但是由于时间和能力的限制, 仍然保留了一部分.
- 我们试着将 `class GEM` 的职责完全抽离开来, 使得 `task_idx` 不作为参数传入. 这需要改动大量的代码, 几乎是重写了 `class GEM` 的所有代码. 我们只在一定程度上实现了这一点, 虽然在效果上和原来的模型几乎无差距, 但是从软件工程项目架构的角度来看, 这一部分仍然需要改进.

4. 复现结果

本次复现使用的环境如下所示

```
PS C:\Users\wuchentian> neofetch
       ,.=:!!t3Z3z.,                wuchentian@YCY
      :tt:::tt333EE3                ---------
      Et:::ztt33EEEL @Ee.,      ..,  OS: Windows 11 x86_64
     ;tt:::tt333EE7 ;EEEEEEttttt33#  Host: LENOVO 82JQ
    :Et:::zt333EEQ. $EEEEEttttt33QL  Kernel: 10.0.22000
    it::::tt333EEF @EEEEEEttttt33F   Uptime: 7 hours, 2 mins
   ;3=*^```"*4EEV :EEEEEEttttt33@.   Packages: 2 (scoop)
   ,.=::::!t=., ` @EEEEEEtttz33QF    Shell: bash 4.4.23
  ;:::::::::zt33)   "4EEEtttji3P*    Resolution: 2560x1600
 :t::::::::tt33.:Z3z..  `` ,..g.     DE: Aero
 i::::::::zt33F AEEEtttt::::ztF      WM: Explorer
;:::::::::t33V ;EEEttttt::::t3       WM Theme: Custom
E::::::::zt33L @EEEttttt::::z3F      Terminal: Windows Terminal
{3=*^```"*4E3) ;EEEttttt:::::tZ`     CPU: AMD Ryzen 7 5800H with Radeon Graphics (16) @ 3.200GHz
            ` :EEEEttttt::::z7       GPU: Caption
              "VEzjt:;;z>*`          GPU: NVIDIA GeForce RTX 3060 Laptop GPU
                                     GPU: AMD Radeon(TM) Graphics
                                     GPU
                                     Memory: 7282MiB / 28524MiB
```

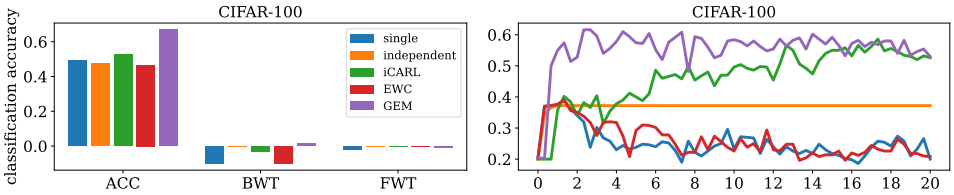
```
(LCL) wuchentian@YCY C:\Users\wuchentian>nvidia-smi
Tue Jan  2 20:30:08 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 527.99       Driver Version: 527.99       CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name            TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ... WDDM  | 00000000:01:00.0 Off |                  N/A |
| N/A   44C    P0    24W /  95W |    336MiB /  6144MiB |      1%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

(LCL) wuchentian@YCY C:\Users\wuchentian>nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Tue_Aug_15_22:09:35_Pacific_Daylight_Time_2023
Cuda compilation tools, release 12.2, V12.2.140
Build cuda_12.2.r12.2/compiler.33191640_0

(LCL) wuchentian@YCY C:\Users\wuchentian>conda -V
conda 4.5.11

(LCL) wuchentian@YCY C:\Users\wuchentian>python -V
Python 3.10.13
```

在原本的 GEM 模型上, 我们经过大概 8 小时的运行, 得到如下的复现结果.



(a) Bar plot of accuracy on CIFAR-100　　(b) Evo plot of accuracy on CIFAR-100

将 GEM 模型嵌入到 LibContinual 中, 我们得到的复现结果如附录B.2所示. 可以发现我们的复现效果基本相当于论文中的效果.

A. 符号说明

符号	含义	注释
$\boldsymbol{x}_i$	特征向量	
$\boldsymbol{y}_i$	目标向量	
$\mathcal{X}$	特征向量集	$\forall i : \boldsymbol{x}_i \in \mathcal{X}$
$\mathcal{Y}$	目标向量集	$\forall i : \boldsymbol{y}_i \in \mathcal{Y}$
D_{tr}	训练集	$D_{\text{tr}} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$
$(\boldsymbol{x}_i, \boldsymbol{y}_i)$	单个样本	$\boldsymbol{x}_i \in \mathcal{X}, y_i \in \mathcal{Y}$
P	概率分布	$(\boldsymbol{x}_i, \boldsymbol{y}_i) \sim P$
f	模型	$f : \mathcal{X} \to \mathcal{Y}$
ℓ	损失函数	
$\|\cdot\|$	模长	
$\mathcal{T}$	任务描述符序列	
$(\boldsymbol{x}_i, t_i, \boldsymbol{y}_i)$	数据序列	$\boldsymbol{x}_i \in \mathcal{X}, t_i \in \mathcal{T}, y_i \in \mathcal{Y}$
$\langle \cdot, \cdot \rangle$	内积	
$\nabla \cdot$	梯度算子	$\nabla_\theta f = \frac{\partial f}{\partial \theta}$

表 1: 符号说明

B. 代码说明

B.1. `git diff`

在终端运行 `git diff main..raw --stat`, 可以展示我们对 LibContinual 进行的修改统计数据.

```
README.lar.md                          | 141 ----------------
README.lyl.md                          |  59 -------
README.md                              |  22 ++--
README.wct.md                          |  25 ---
README.ycy.md                          |  34 ----
config/backbones/resnet18.yaml         |   7 -
config/finetune.yaml                   |   2 +-
config/gem.yaml                        |  60 -------
config/headers/device.yaml             |   2 +-
config/icarl.yaml                      |   4 +-
config/lucir.yaml                      |  25 +--
config/lucir2.yaml                     |  63 -------
config/lwf.yaml                        |   4 +-
core/data/__init__.py                  |   2 +-
core/data/augments.py                  |  65 ++++++-
```

```
core/data/cutout.py                          |   5 +
core/data/dataloader.py                      |  99 ++---------
core/data/dataset.py                         |  89 ++--------
core/model/__init__.py                       |   2 +-
core/model/backbone/__init__.py              |   2 +-
core/model/backbone/resnet.py                | 161 +++++++----------
core/model/buffer/__init__.py                |   1 -
core/model/buffer/linearbuffer.py            |   8 +-
core/model/buffer/linearherdingbuffer.py     |   3 +
core/model/buffer/ringbuffer.py              |  11 --
core/model/buffer/update.py                  |   3 +-
core/model/replay/__init__.py                |   3 +-
core/model/replay/common.py                  |  80 ---------
core/model/replay/ewc.py                     |   0
core/model/replay/finetune.py                |  25 +--
core/model/replay/gem.py                     | 195 ----------------------
core/model/replay/icarl.py                   |  82 +++++-----
core/model/replay/lucir.py                   |  19 ++-
core/model/replay/lwf.py                     |   3 +
core/result.txt                              |   0
core/test.py                                 |  37 ----
core/trainer.py                              | 279
   ++++++++++++-------------------
core/utils/logger.py                         |   2 +-
core/utils/metrics.py                        |  56 -------
core/utils/utils.py                          |  64 ++-----
config/ewc.yaml => log/.gitkeep              |   0
log/log                                      |   0
run_trainer.py                               |   8 +-
44 files changed, 432 insertions(+), 1327 deletions(-)
```

B.2. log file

下面展示了我们的日志, 由于我们的可视化输出使用的是 `tqdm`, 因此进度条本身没有被记录在日志中, 只有结果保留在日志中了. 日志的部分截图如图 3 所示

图 3: 日志部分截图

```
task num:  20
task num:  20
GEM(
  (net): ResNet4GEM(
    (conv1): Conv2d(3, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1,
     1), bias=False)
    (bn1): BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
    (layer1): Sequential(
      (0): BasicBlock4GEM(
        (conv1): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1),
    padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
        (conv2): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1),
    padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
        (shortcut): Sequential()
      )
      (1): BasicBlock4GEM(
        (conv1): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1),
    padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
        (conv2): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1),
    padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
        (shortcut): Sequential()
      )
    )
    (layer2): Sequential(
      (0): BasicBlock4GEM(
        (conv1): Conv2d(20, 40, kernel_size=(3, 3), stride=(2, 2),
    padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(40, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
        (conv2): Conv2d(40, 40, kernel_size=(3, 3), stride=(1, 1),
    padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(40, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
        (shortcut): Sequential(
          (0): Conv2d(20, 40, kernel_size=(1, 1), stride=(2, 2), bias=
    False)
          (1): BatchNorm2d(40, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
```

```
      )
    )
    (1): BasicBlock4GEM(
      (conv1): Conv2d(40, 40, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(40, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(40, 40, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(40, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer3): Sequential(
    (0): BasicBlock4GEM(
      (conv1): Conv2d(40, 80, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(80, 80, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(40, 80, kernel_size=(1, 1), stride=(2, 2), bias=
False)
        (1): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock4GEM(
      (conv1): Conv2d(80, 80, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(80, 80, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer4): Sequential(
    (0): BasicBlock4GEM(
      (conv1): Conv2d(80, 160, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
```

```
        (bn1): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
        (conv2): Conv2d(160, 160, kernel_size=(3, 3), stride=(1, 1),
    padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
        (shortcut): Sequential(
          (0): Conv2d(80, 160, kernel_size=(1, 1), stride=(2, 2), bias=
    False)
          (1): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
        )
      )
      (1): BasicBlock4GEM(
        (conv1): Conv2d(160, 160, kernel_size=(3, 3), stride=(1, 1),
    padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
        (conv2): Conv2d(160, 160, kernel_size=(3, 3), stride=(1, 1),
    padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
        (shortcut): Sequential()
      )
    )
    (linear): Linear(in_features=160, out_features=100, bias=True)
  )
  (ce): CrossEntropyLoss()
)
Trainable params in the model: 1109240
================Task 0 Start!================
================Task 1 Start!================
================Task 2 Start!================
================Task 3 Start!================
================Task 4 Start!================
================Task 5 Start!================
================Task 6 Start!================
================Task 7 Start!================
================Task 8 Start!================
================Task 9 Start!================
================Task 10 Start!================
================Task 11 Start!================
================Task 12 Start!================
================Task 13 Start!================
================Task 14 Start!================
================Task 15 Start!================
================Task 16 Start!================
```

```
================Task 17 Start!================
================Task 18 Start!================
================Task 19 Start!================
Final Accuracy: 0.7092
Backward: -0.0760
Forward:  -0.0106
Time cost :  35285.64979815483
```

参考文献

[1] David Lopez-Paz and Marc'Aurelio Ranzato. *Gradient Episodic Memory for Continual Learning*. Sept. 2022. arXiv: 1706.08840 [cs]. (Visited on 09/28/2023).

[2] Michael McCloskey and Neal J. Cohen. "Catastrophic interference in connectionist networks: The sequential learning problem". In: ed. by Gordon H. Bower. Vol. 24. Psychology of learning and motivation. Academic Press, 1989, pp. 109–165. doi: 10.1016/S0079-7421(08)60536-8.

[3] Andrei A. Rusu et al. *Progressive Neural Networks*. Oct. 2022. arXiv: 1606.04671 [cs]. (Visited on 11/29/2023).

[4] Chrisantha Fernando et al. *PathNet: Evolution Channels Gradient Descent in Super Neural Networks*. Jan. 2017. eprint: 1701.08734 (cs). (Visited on 11/29/2023).

[5] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[6] Sylvestre-Alvise Rebuffi et al. "iCaRL: Incremental Classifier and Representation Learning". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, July 2017, pp. 5533–5542. isbn: 978-1-5386-0457-1. doi: 10.1109/CVPR.2017.587. (Visited on 11/10/2023).

[7] Abhishek Aich. *Elastic Weight Consolidation (EWC): Nuts and Bolts*. May 2021. arXiv: 2105.04093 [cs, stat]. (Visited on 11/19/2023).

[8] Saihui Hou et al. "Learning a Unified Classifier Incrementally via Rebalancing". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: IEEE, June 2019, pp. 831–839. isbn: 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00092. (Visited on 11/10/2023).

[9] James Kirkpatrick et al. "Overcoming Catastrophic Forgetting in Neural Networks". In: *Proceedings of the National Academy of Sciences* 114.13 (Mar. 2017), pp. 3521–3526. issn: 0027-8424, 1091-6490. doi: 10.1073/pnas.1611835114. arXiv: 1612.00796 [cs, stat]. (Visited on 11/12/2023).

[10] Xiang Li et al. "Adversarial Multimodal Representation Learning for Click-Through Rate Prediction". In: *Proceedings of The Web Conference 2020*. Apr. 2020, pp. 827–836. doi: 10.1145/3366423.3380163. arXiv: 2003.07162 [cs, stat]. (Visited on 11/19/2023).