人工智能中的编程第五次作业

本作业旨在通过实现一个简单的深度学习框架来加深对深度学习原理和算法的理解并熟悉一些Python的代码风格。我们将模仿 Needle 框架，分成四个部分完成：运算符的前向运算、梯度计算、拓扑排序以及Tensor类的自动微分以实现一个完整的数据结构。

核心概念: 本次实验的核心是构建一个计算图。每个 `Tensor` 对象都是图中的一个节点（`Value`）。当两个 `Tensor`（例如 `A` 和 `B`）通过一个操作（例如 `+`）产生一个新的 `Tensor`（`C`）时，我们实际上创建了一个 `Add` 操作（`Op`），并将 `A` 和 `B` 作为其输入。`C` 会记录这个 `Add` 操作和它的输入，从而形成一个有向无环图（DAG）。自动微分就是在这个图上反向传播梯度。

1. 运算符的前向运算 (`task1_operators.py`)

在这一部分，你需要补充和完善Tensor类，使其能够执行基础的前向运算。Tensor类已经在 `task1_operators.py` 中被定义，但缺少一些核心的功能。你的任务是为每个 `Op` 子类实现 `compute` 方法，完成前向计算逻辑。

实现步骤：

- 阅读 `basic_operator.py`，理解 `Op` 类和 `Value` 类的基本结构和功能。`Value` 是计算图中的节点，`Op` 是连接节点的操作。
- 在 `task1_operators.py` 中，找到各个运算符类（如 `EWiseAdd`，`MatMul` 等）。
- 为每个运算符类实现 `compute` 方法。这个方法接收 `np.ndarray` 作为输入，并应返回一个 `np.ndarray` 作为计算结果。例如，`MatMul.compute` 应该执行 `np.matmul`。
- 实现后，可以利用 `test_task1_forward.py` 中的测试函数进行验证。

2. 梯度的反向计算 (`task1_operators.py`)

本部分的目标是为 `task1_operators.py` 中定义的每个 `Op` 实现梯度计算功能。你需要为每个运算符补充其 `gradient` 方法。

实现步骤：

- 梯度计算的本质是链式法则。`gradient` 方法接收上游传来的梯度 `out_grad` 和当前节点 `node`，需要计算并返回针对其每个输入的梯度。
- 梯度也应以 `Tensor` 对象的形式表示。例如，对于加法 `C = A + B`，`C` 的梯度会原封不动地传给 `A` 和 `B`，所以 `Add.gradient` 应该返回 `(out_grad, out_grad)`。对于矩阵乘法 `C = A @ B`，A的梯度是 `out_grad @ B.T`，B的梯度是 `A.T @ out_grad`。
- 注意处理好广播和维度变换操作（如 `Summation`、`BroadcastTo`）的梯度。
- 可以利用 `test_task1_backward.py` 中的 `gradient_check` 函数来数值化地检验你的梯度实现是否正确。

3. 拓扑排序 (`task2_auto_diff.py`)

在反向传播时，我们需要保证一个节点的梯度被计算之前，所有依赖于它的节点的梯度都已经被计算。拓扑排序为此提供了正确的计算顺序。

实现步骤：

- 在 `task2_auto_diff.py` 中，你需要实现 `find_topo_sort` 函数。
- 该函数接收一个包含最终输出节点的列表 `node_list` 。
- 你需要从这些输出节点开始，深度优先遍历（DFS）整个计算图，并在回溯时（post-order）将节点加入排序列表。这样可以确保所有输入节点排在输出节点之前。
- 完成后，可以利用 `test_task2_topo_sort.py` 进行测试。

4. Tensor类的自动微分 (`task2_auto_diff.py` ， `tensor.py`)

最后一部分是将前面的内容整合，完成一个能够进行自动微分的 `TensorFull` 类。这需要你实现 `compute_gradient_of_variables` 函数，并在 `TensorFull` 中调用它。

实现步骤：

- 在 `task2_autodiff.py` 中，实现 `compute_gradient_of_variables` 函数。该函数是自动微分的核心。
- **函数逻辑**:
  1. 调用 `find_topo_sort` 获取计算图的拓扑排序。
  2. 初始化一个字典，用于存储每个节点的梯度。将输出节点 `output_node` 的梯度初始化为 `out_grad` 。
  3. **反向**遍历拓扑排序列表。对于每个节点：
     a. 如果该节点是叶子节点（即用户创建的 `Tensor` ， `op` 为 `None` ），则跳过。
     b. 获取当前节点的梯度。
     c. 调用当前节点的 `op.gradient()` 方法，计算其对其输入的梯度。
     d. 将计算出的梯度累加到相应输入节点的梯度中。注意，一个节点可能被多次用作输入，所以梯度需要累加。
- 在 `tensor.py` 中， `TensorFull` 的 `backward` 方法已经为你准备好。它会调用你实现的 `compute_gradient_of_variables` 。
- 完成后，可以利用 `test_task2_auto_diff.py` 中的测试进行验证。

其他文件

- 基础环境： `basic_operator.py` 中定义了 `Op` 类和 `Value` 类，这是整个框架的基础。在 `device.py` 中定义了模拟的CPU设备。
- 工具函数： 在 `utils.py` 中提供了一些可能会用到的工具函数。为了避免交叉引用，可以根据需要将这些函数复制到你的代码中。
- 测试文件：以 `test_` 开头。在每完成一个部分后，可以使用相应的测试文件进行验证，确保每一步的实现都是正确的。

作业提交

- 本次作业总分 10 分，会根据完成情况按部分给分。同时请按照作业编排顺序完成，请对于微分图进行显式构建以完成自动微分，给出的测试文件只是参考。我们会根据代码实现进行评分。
- 请于2025.11.30 晚 23 时 59 分前于 course.pku.edu.cn 提交代码和一个简短的报告说明