

人工智能中的编程——第二次作业

问题描述

- 实现全连接层的正向传播和反向传播
- 实现卷积层的正向传播和反向传播
 - 实现 `im2col` 与 `col2im`
- 实现 `max_pool_forward` 与 `max_pool_backward`
- 实现 `softmax_forward`
- 实现 `cross_entropy_forward` 与 `cross_entropy_backward` (with `softmax`)

实现

本轮作业的实现主要在 `csrc/layers/` 与 `csrc/core/kernels` 中。

我的实现分为两个层面：

- 在 `csrc/core/kernels` 中实现指针层面的操作，定义模板结构体来实现不同设备异构计算的统一，可参考 `csrc/core/kernels/ops.h` 与其对应的 `cpp`、`cu` 文件，还可阅读 `csrc/core/kernels/functions` 文件夹内代码
- 在 `csrc/layers/` 中实现张量层面的操作，可参考 `csrc/layers/layers.h` 与其对应的 `cpp` 文件，该层面中调用 `csrc/core/kernels` 中的模板结构体
 - 由于不同的设备的异构计算算子在第一层抽象中已经统一，因此在该层面中不需要再通过 `cpp` 与 `cu` 文件来区分设备类型，仅需使用 `tensor` 类管理设备类型，并调用 `csrc/core/kernels` 中的模板结构体即可

全连接层

签名在 `csrc/layers/layers.h` 中定义：

```
1  /**
2   * @brief forward function for fully connected layer
3   *      -  $Y = XW + b$ 
4   */
5  template <typename Tp>
6  void fc_forward(
7      const tensor::Tensor<Tp>& input,      // x(batch_size, in_features)
8      const tensor::Tensor<Tp>& weight,      // w(in_features, out_features)
9      const tensor::Tensor<Tp>& bias,        // b(out_features)
10     tensor::Tensor<Tp>& output             // y(batch_size, out_features)
11 );
12
13 /**
14 * @brief backward function for fully connected layer
15 *      -  $dx = dy * w^T$ 
16 *      -  $dw = x^T * dy$ 
17 *      -  $db = \sum dy$ 
18 */
19 template <typename Tp>
20 void fc_backward(
```

```

21     const tensor::Tensor<Tp>& input,           // X(batch_size, in_features)
22     const tensor::Tensor<Tp>& weight,          // W(in_features, out_features)
23     const tensor::Tensor<Tp>& bias,             // b(1, out_features)
24     const tensor::Tensor<Tp>& output,           // Y(batch_size, out_features)
25     tensor::Tensor<Tp>& grad_input,            // dX(batch_size, in_features)
26     tensor::Tensor<Tp>& grad_weight,           // dW(in_features, out_features)
27     tensor::Tensor<Tp>& grad_bias,             // dB(1, out_features)
28     const tensor::Tensor<Tp>& grad_output      // dY(batch_size, out_features)
29 );

```

函数实现在 `csrc/layers/fc_layer.cpp` 中，测试在 `csrc/layers/tests/test_layer.cpp` 中

测试样例使用 pytorch 生成，生成脚本保存在 `csrc/layers/tests/` 文件夹中

卷积层

签名在 `csrc/layers/layers.h` 中定义：

```

1  /**
2   * @brief forward function for conv2d layer
3   *      -  $Y = W \text{ conv } X + b$ 
4   */
5  template <typename Tp>
6  void conv2d_forward(
7      const tensor::Tensor<Tp>& input,           // X(batch_size, in_channels,
8      height, width)
9      const tensor::Tensor<Tp>& weight,          // W(out_channels, in_channels,
10     kernel_h, kernel_w)
11     const tensor::Tensor<Tp>& bias,             // b(out_channels)
12     tensor::Tensor<Tp>& output,                 // Y(batch_size, out_channels,
13     height_out, width_out)
14     const int pad_h,
15     const int pad_w,
16     const int stride_h,
17     const int stride_w
18 );
19
20 /**
21  * @brief backward function for conv2d layer
22  *      -  $dX = dY \text{ conv } W^T$ 
23  *      -  $dW = dY \text{ conv } X$ 
24  *      -  $dB = \sum dY$ 
25  */
26 template <typename Tp>
27 void conv2d_backward(
28     const tensor::Tensor<Tp>& input,           // X(batch_size, in_channels,
29     height, width)
30     const tensor::Tensor<Tp>& weight,          // W(out_channels, in_channels,
31     kernel_h, kernel_w)
32     tensor::Tensor<Tp>& grad_input,            // dX(batch_size, in_channels,
33     height, width)
34     tensor::Tensor<Tp>& grad_weight,           // dW(out_channels, in_channels,
35     kernel_h, kernel_w)
36     tensor::Tensor<Tp>& grad_bias,             // dB(1, out_channels)

```

```

30     const tensor::Tensor<Tp>& grad_output, // dY(batch_size, out_channels,
height_out, width_out)
31     const int pad_h,
32     const int pad_w,
33     const int stride_h,
34     const int stride_w
35 );

```

函数实现在 `csrc/layers/conv2d_layer.cpp` 中，测试在 `csrc/layers/tests/test_layer.cpp` 中其指针层面计算在 `csrc/core/kernels/ops.h` 中定义，可参考 `csrc/core/kernels/ops.h` 及其对应的 `cpp`、`cu` 文件

im2col 与 col2im

签名在 `csrc/core/kernels/ops.h` 中定义，可参考 `csrc/core/kernels/ops.h` 及其对应的 `cpp`、`cu` 文件

对应测试在 `csrc/core/kernels/tests/test_ops_cpu.cpp` 与 `csrc/core/kernels/tests/test_ops_gpu.cu` 中

测试样例使用 `pytorch` 生成，生成脚本及对应样例保存在 `csrc/core/kernels/tests/` 文件夹中

池化层

签名在 `csrc/layers/layers.h` 中定义：

```

1  /**
2   * @brief forward function for max pooling layer
3   */
4  template <typename Tp>
5  void max_pool_forward(
6      const tensor::Tensor<Tp>& input, // X(batch_size, channels, height,
width)
7      tensor::Tensor<int>& mask, // mask(batch_size, channels,
height_out, width_out)
8      tensor::Tensor<Tp>& output, // Y(batch_size, channels,
height_out, width_out)
9      const int kernel_h,
10     const int kernel_w,
11     const int pad_h,
12     const int pad_w,
13     const int stride_h,
14     const int stride_w
15 );
16
17 /**
18  * @brief backward function for max pooling layer
19  */
20 template <typename Tp>
21 void max_pool_backward(
22     tensor::Tensor<Tp>& grad_input, // dX(batch_size, channels,
height, width)
23     const tensor::Tensor<int>& mask, // mask(batch_size, channels,
height_out, width_out)

```

```

24     const tensor::Tensor<Tp>& grad_output, // dY(batch_size, channels,
height_out, width_out)
25     const int kernel_h,
26     const int kernel_w,
27     const int pad_h,
28     const int pad_w,
29     const int stride_h,
30     const int stride_w
31 );

```

函数实现在 `csrc/layers/pooling_layer.cpp` 中，测试在 `csrc/layers/tests/test_layer.cpp` 中
其指针层面计算在 `csrc/core/kernels/ops.h` 中定义，可参考 `csrc/core/kernels/pooling_ops.h`
及其对应的 `cpp`、`cu` 文件

测试样例使用 `pytorch` 生成，生成脚本及对应样例保存在 `csrc/layers/tests/` 文件夹中

softmax层

签名在 `csrc/layers/layers.h` 中定义：

```

1  /**
2   * @brief forward function for softmax layer
3   *      - Y = softmax(X)
4   */
5  template <typename Tp>
6  void softmax_forward(
7      const tensor::Tensor<Tp>& input,      // X(batch_size, num_classes)
8      tensor::Tensor<Tp>& output            // Y(batch_size, num_classes)
9  );

```

函数实现在 `csrc/layers/softmax_layer.cpp` 中，测试在 `csrc/layers/tests/test_layer.cpp` 中
其指针层面计算在 `csrc/core/kernels/functions/softmax.h` 中定义，可参考
`csrc/core/kernels/functions/softmax.h` 及其对应的 `cpp`、`cu` 文件

对应测试在 `csrc/core/kernels/tests/test_functions_cpu.cpp` 与
`csrc/core/kernels/tests/test_functions_gpu.cu` 中

测试样例使用 `pytorch` 生成，生成脚本保存在 `csrc/core/functions/tests` 文件夹中

cross_entropy层

签名在 `csrc/layers/layers.h` 中定义：

```

1  /**
2   * @brief loss function for cross entropy
3   *      - loss = -\sum y_i * log(p_i)
4   */
5  template <typename Tp>
6  void cross_entropy_forward(
7      const tensor::Tensor<Tp>& input,      // X(batch_size, num_classes)
8      const tensor::Tensor<int>& target,    // t(batch_size)
9      tensor::Tensor<Tp>& output            // z(1)
10 );

```

```

11
12  /**
13   * @brief backward function for cross entropy
14   *       - dx_i = p_i - y_i
15   */
16  template <typename Tp>
17  void cross_entropy_backward(
18      const tensor::Tensor<Tp>& input,    // x(batch_size, num_classes)
19      const tensor::Tensor<int>& target,  // t(batch_size)
20      tensor::Tensor<Tp>& grad            // dx(batch_size, num_classes)
21  );

```

函数实现在 `csrc/layers/softmax_layer.cpp` 中，测试在 `csrc/layers/tests/test_layer.cpp` 中

其指针层面计算在 `csrc/core/kernels/functions/cross_entropy.h` 中定义，可参考

`csrc/core/kernels/functions/cross_entropy.h` 及其对应的 `cpp`、`cu` 文件

对应测试在 `csrc/core/kernels/tests/test_functions_cpu.cpp` 与

`csrc/core/kernels/tests/test_functions_gpu.cu` 中

测试样例使用 `pytorch` 生成，生成脚本保存在 `csrc/core/functions/tests` 文件夹中

测试方法

本项目使用 `CMake` 构建，使用 `Google Test` 进行单元测试，我已经为项目内**所有**模块编写了单元测试，

`CPU` 代码实现可以通过 `GitHub Action` 自动测试，也可以手动编译进行测试。

可进入根目录下的 `scripts` 文件夹，运行：

```
1 | bash all_test.sh
```

若想分别测试 `cpu` 与 `gpu` 版本，可进入 `scripts` 文件夹，运行

```
1 | bash test.sh --cpu
2 | bash test.sh --gpu
```

若使用 `windows` 系统进行测试，按照下面步骤：

编译

```

1 | mkdir build
2 | cd build
3 | cmake -DTEST=ON -DCUDA=OFF ..
4 | make

```

- `-DTEST`：是否开启测试，默认为 `OFF`
- `-DCUDA`：是否开启 `CUDA` 支持，默认为 `OFF`

测试

编译后请进入 `build` 目录，执行：

```
1 | ctest --verbose --output-on-failure -C Debug -T test
```

如需进行 GPU 代码测试，请重新编译：

```
1 | cmake -DTEST=ON -DCUDA=ON ..  
2 | make  
3 | ctest --verbose --output-on-failure -C Debug -T test
```