BangC 编程实践

使用 BangC 编程完成下述 **4** 个操作。
在测试输出结果时，可采用与对应的 cpu 实现对比精度标准有 Mean absolute error(MAE)和 average relative error(ARE)等。

$$\mathrm{MAE}=\frac{\sum_{i=1}^{n}|CPU_{result}-MLU_{reslut}|}{n}=\frac{\sum_{i=1}^{n}|ei|}{n}$$

$$ARE=\frac{\sum_{i=1}^{n}|CPU_{result}-MLU_{result}|}{\sum_{i=1}^{n}|CPU_{result}|}$$

1. Softmax

问题描述
利用指数化归一函数使得每一列和为 1。
参考公式如下：

$$soft\max(x)_i=\frac{e^{x_i}}{\sum_j e^{x_j}}$$

通常为了避免输入较大导致指数化数值溢出，需要将所有的输入数值控制在 0 及 0 以下，即：

$$soft\max(x)_i=\frac{e^{x_i-\max}}{\sum_j e^{x_j-\max}}$$

给定输入 m x n(20 x 256)规模的矩阵 X，对每一列利用指数化归一函数，输出 m x n 的归一化矩阵。

操作步骤
1）补全~/Cambricon_MLU270_Test/BangC/practice/bangSoftmax/kernel_softmax.mlu 文件中的 SoftmaxKernel()函数
2）在~/Cambricon_MLU270_Test/BangC/practice/bangSoftmax/路径下执行 make ;编译程序
3）运行 ./test ；执行完成后会打印 MLU 上操作的硬件时间和与 cpu 的精度误差

2. Cosine 相似度

问题描述
Cosine 相似度是一种相似性度量，输出范围为-1 到+1,0 代表无相关性，负值代表负相关，正值代表正相关。请实现向量间的余弦相似度计算。
参考公式如下：

$$\mathrm{c}(X,Y)=\frac{X\cdot Y}{|X||Y|}=\frac{\sum_{i=1}^{n}X_iY_i}{\sqrt{\sum_{i=1}^{n}X_i^{\,2}}\sqrt{\sum_{i=1}^{n}Y_i^{2}}}$$

给定输入 m x n(256x256)规模的矩阵 X 和 Y，按对应列求余弦相似度，输出 1 x n 的余弦相似度矩阵。
建议使用 MAE 对比精度。
操作步骤
1）补全~/Cambricon_MLU270_Test/BangC/practice/bangCosine/kernel_cosine.mlu 文件中的 CosineKernel()函数
2）在~/Cambricon_MLU270_Test/BangC/practice/bangCosine/路径下执行 make ;编译程序
3）运行 ./test ；执行完成后会打印 MLU 上操作的硬件时间和与 cpu 的精度误差

3. Batch Normalization

问题描述
神经网络在训练过程中，前一层权重参数的改变会造成每层输入样本分布的改变，这造成了训练过程的困难。为了解决这个问题，通常会使用小的学习率和参数初始化技巧，就导致了训练速度变慢，尤其是训练具有饱和非线性的模型时。我们将这一现象定义为 internal covariate shift，并提出通过规范化输入来解决。
给定输入 m x n (128 x 256)规模的矩阵 B，逐行求 BN，输出 m x n 标准化后的矩阵。
参考步骤：
1. 求出平均值

$$\mu_{\mathrm{B}}=\frac{1}{\mathrm{n}}\sum_{\mathrm{i}=1}^{n}x_i$$

2. 求方差

$$\sigma_{\mathrm{B}}^{2}=\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu_B)^2$$

3. 标准化

$$\hat{x}_i=\frac{x_i-\mu_B}{\sqrt{\sigma_B^2+\varepsilon}}$$

4. 缩放和平移

$$\mathrm{y}_{\mathrm{i}}=\gamma\hat{x}+\beta$$

操作步骤：
1）补全~/Cambricon_MLU270_Test/BangC/practice/bangBN/kernel_bn.mlu 文件中的 bnKernel()函数
2）在~/Cambricon_MLU270_Test/BangC/practice/bangBN/路径下执行 make ;编译程序
3）运行 ./test ；执行完成后会打印 MLU 上操作的硬件时间和与 cpu 的精度误差

4. Triplet Loss

问题描述
Triplet Losss 的核心是锚例、正例、负例共享模型，通过模型将锚例与正例聚类，远离负例。参考公式如下：

$$\mathrm{L} = \max(d(a,p) - d(a,n) + m\arg in, 0)$$

这里，我们指定 d 为曼和顿距离（Manhattan Distinct），即两点间对应坐标分量误差的绝对值之和：

$$\mathrm{d}(X,Y) = \sum_{i=1}^{n} |X_i + Y_i|$$

最终的优化目标是拉近 a，p 的距离；拉远 a，n 的距离。

操作步骤：

1）补全~/Cambricon_MLU270_Test/BangC/practice/bangTripletloss/kernel_tripletloss.mlu 文件中的 TripletlossKernel()函数

2）在~/Cambricon_MLU270_Test/BangC/practice/bangTripletloss/路径下执行 make ;编译程序

3）运行 ./test ；执行完成后会打印 MLU 上操作的硬件时间和与 cpu 的精度误差