d:; cd 'd:\RA data\dlrm-main\DLRM\_memory\_trace'; & 'c:\Users\White\AppData\Local\Programs\Python\Python39\python.exe' 'D:\RA data\dlrm-main\DLRM\_memory\_trace\dlrm\_s\_pytorch.py' --mini-batch-size=2 --data-size=4 --arch-embedding-size=6

d:; cd 'd:\RA data\dlrm-main'; & 'c:\Users\White\AppData\Local\Programs\Python\Python39\python.exe' 'D:\RA data\dlrm-main\dlrm\_s\_pytorch.py' --mini-batch-size=2 --data-size=2 --debug-mode --arch-embedding-size=6

X, lS\_o, lS\_i, T, W, CBPP = unpack\_batch(inputBatch)

$ python dlrm\_s\_pytorch.py --mini-batch-size=2 --data-size=6 --debug-mode

model arch:

mlp top arch 3 layers, with input to output dimensions:

[8 4 2 1]

# of interactions

8

mlp bot arch 2 layers, with input to output dimensions:

[4 3 2]

# of features (sparse and dense)

4

dense feature size

4

sparse feature size

2

# of embeddings (= # of sparse features) 3, with dimensions 2x:

[4 3 2]

data (inputs and targets):

mini-batch: 0

[[0.69647 0.28614 0.22685 0.55131]

[0.71947 0.42311 0.98076 0.68483]]

**密集特征矩阵** (**X**)：这是一个形状为**(mini-batch-size, dense feature size)**的张量，其中包含了每个样本的密集特征。例如，在**mini-batch: 0**中，**[[0.69647 0.28614 0.22685 0.55131] [0.71947 0.42311 0.98076 0.68483]]**表示两个样本，每个样本有4个密集特征。

[[[1], [0, 1]], [[0], [1]], [[1], [0]]]

**稀疏特征索引** (**lS\_i**) 和 **长度（或偏移）** (**lS\_o**)：这些是用于处理稀疏特征的索引和长度（或偏移量），用于从嵌入表中检索特定的嵌入向量。例如，**[[[1], [0, 1]], [[0], [1]], [[1], [0]]]**表示有多个稀疏特征字段，每个字段有不同的索引列表。

[[0.55679]

[0.15896]]

**目标值** (**T**)：这是一个形状为**(mini-batch-size, 1)**的张量，包含了每个样本的目标值（例如点击与否）。在**mini-batch: 0**中，**[[0.55679] [0.15896]]**表示两个样本的目标值。

mini-batch: 1

[[0.36179 0.22826 0.29371 0.63098]

[0.0921 0.4337 0.43086 0.49369]]

[[[1], [0, 2, 3]], [[1], [1, 2]], [[1], [1]]]

[[0.15307]

[0.69553]]

mini-batch: 2

[[0.60306 0.54507 0.34276 0.30412]

[0.41702 0.6813 0.87546 0.51042]]

[[[2], [0, 1, 2]], [[1], [2]], [[1], [1]]]

[[0.31877]

[0.69197]]

**初始参数（weights and bias）**

initial parameters (weights and bias):

[[ 0.05438 -0.11105]

[ 0.42513 0.34167]

[-0.1426 -0.45641]

[-0.19523 -0.10181]]

[[ 0.23667 0.57199]

[-0.16638 0.30316]

[ 0.10759 0.22136]]

[[-0.49338 -0.14301]

[-0.36649 -0.22139]]

[[0.51313 0.66662 0.10591 0.13089]

[0.32198 0.66156 0.84651 0.55326]

[0.85445 0.38484 0.31679 0.35426]]

[0.17108 0.82911 0.33867]

[[0.55237 0.57855 0.52153]

[0.00269 0.98835 0.90534]]

[0.20764 0.29249]

[[0.52001 0.90191 0.98363 0.25754 0.56436 0.80697 0.39437 0.73107]

[0.16107 0.6007 0.86586 0.98352 0.07937 0.42835 0.20454 0.45064]

[0.54776 0.09333 0.29686 0.92758 0.569 0.45741 0.75353 0.74186]

[0.04858 0.7087 0.83924 0.16594 0.781 0.28654 0.30647 0.66526]]

[0.11139 0.66487 0.88786 0.69631]

[[0.44033 0.43821 0.7651 0.56564]

[0.0849 0.58267 0.81484 0.33707]]

[0.92758 0.75072]

[[0.57406 0.75164]]

[0.07915]

### 模型架构（DLRM\_Net）

* **嵌入层** (**emb\_l**): 使用**EmbeddingBag**表示，处理稀疏特征。**EmbeddingBag(4, 2, mode=sum)**表示一个嵌入表，有4个嵌入向量，每个向量的维度为2，使用求和模式聚合。
* **底部MLP** (**bot\_l**): 用于处理输入的密集特征。**Linear(in\_features=4, out\_features=3, bias=True)**表示一个线性层，将输入的4维特征转换为3维。
* **顶部MLP** (**top\_l**): 处理合并后的特征（经过特征交互）。**Linear(in\_features=8, out\_features=4, bias=True)**表示一个线性层，将8维的输入特征转换为4维。

DLRM\_Net(

(emb\_l): ModuleList(

(0): EmbeddingBag(4, 2, mode=sum)

(1): EmbeddingBag(3, 2, mode=sum)

(2): EmbeddingBag(2, 2, mode=sum)

)

(bot\_l): Sequential(

(0): Linear(in\_features=4, out\_features=3, bias=True)

(1): ReLU()

(2): Linear(in\_features=3, out\_features=2, bias=True)

(3): ReLU()

)

(top\_l): Sequential(

(0): Linear(in\_features=8, out\_features=4, bias=True)

(1): ReLU()

(2): Linear(in\_features=4, out\_features=2, bias=True)

(3): ReLU()

(4): Linear(in\_features=2, out\_features=1, bias=True)

(5): Sigmoid()

)

)

time/loss/accuracy (if enabled):

Finished training it 1/3 of epoch 0, -1.00 ms/it, loss 0.451893, accuracy 0.000%

Finished training it 2/3 of epoch 0, -1.00 ms/it, loss 0.402002, accuracy 0.000%

Finished training it 3/3 of epoch 0, -1.00 ms/it, loss 0.275460, accuracy 0.000%

updated parameters (weights and bias):

[[ 0.0543 -0.1112 ]

[ 0.42513 0.34167]

[-0.14283 -0.45679]

[-0.19532 -0.10197]]

[[ 0.23667 0.57199]

[-0.1666 0.30285]

[ 0.10751 0.22124]]

[[-0.49338 -0.14301]

[-0.36664 -0.22164]]

[[0.51313 0.66663 0.10591 0.1309 ]

[0.32196 0.66154 0.84649 0.55324]

[0.85444 0.38482 0.31677 0.35425]]

[0.17109 0.82907 0.33863]

[[0.55238 0.57857 0.52154]

[0.00265 0.98825 0.90528]]

[0.20764 0.29244]

[[0.51996 0.90184 0.98368 0.25752 0.56436 0.807 0.39437 0.73107]

[0.16096 0.60055 0.86596 0.98348 0.07938 0.42842 0.20453 0.45064]

[0.5476 0.0931 0.29701 0.92752 0.56902 0.45752 0.75351 0.74187]

[0.04849 0.70857 0.83933 0.1659 0.78101 0.2866 0.30646 0.66526]]

[0.11137 0.66482 0.88778 0.69627]

[[0.44029 0.43816 0.76502 0.56561]

[0.08485 0.5826 0.81474 0.33702]]

[0.92754 0.75067]

[[0.57379 0.7514 ]]

[0.07908]

d PS D:\RA data\dlrm-main> d:; cd 'd:\RA data\dlrm-main'; & 'c:\Users\White\AppData\Local\Programs\Python\Python39\python.exe' 'D:\RA data\dlrm-main\dlrm\_s\_pytorch.py' --mini-batch-size=2 --data-size=6 --debug-mode

Unable to import mlperf\_logging, No module named 'mlperf\_logging'

2024-03-06 13:56:51.475226: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF\_ENABLE\_ONEDNN\_OPTS=0`.

WARNING:tensorflow:From C:\Users\White\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse\_softmax\_cross\_entropy is deprecated. Please use tf.compat.v1.losses.sparse\_softmax\_cross\_entropy instead.

Unable to import onnx. No module named 'onnx'

Using CPU...

model arch:

mlp top arch 3 layers, with input to output dimensions:

[8 4 2 1]

# of interactions

8

mlp bot arch 2 layers, with input to output dimensions:

[4 3 2]

# of features (sparse and dense)

4

dense feature size

4

sparse feature size

2

# of embeddings (= # of sparse features) 3, with dimensions 2x:

[4 3 2]

data (inputs and targets):

mini-batch: 0

tensor([[0.6965, 0.2861, 0.2269, 0.5513],

[0.7195, 0.4231, 0.9808, 0.6848]])

**密集特征矩阵** (**X**)：这是一个形状为**(mini-batch-size, dense feature size)**的张量，其中包含了每个样本的密集特征。例如，在**mini-batch: 0**中，**[[0.69647 0.28614 0.22685 0.55131] [0.71947 0.42311 0.98076 0.68483]]**表示两个样本，每个样本有4个密集特征。

tensor([[1, 2],

[1, 1],

[1, 1]], dtype=torch.int32)

[tensor([1, 0, 1]), tensor([0, 1]), tensor([1, 0])]

**稀疏特征索引** (**lS\_i**) 和 **长度（或偏移）** (**lS\_o**)：这些是用于处理稀疏特征的索引和长度（或偏移量），用于从嵌入表中检索特定的嵌入向量。例如，**[[[1], [0, 1]], [[0], [1]], [[1], [0]]]**表示有多个稀疏特征字段，每个字段有不同的

tensor([[0.3618],

[0.2283]])

**目标值** (**T**)：这是一个形状为**(mini-batch-size, 1)**的张量，包含了每个样本的目标值（例如点击与否）。在**mini-batch: 0**中，**[[0.55679] [0.15896]]**表示两个样本的目标值。

mini-batch: 1

tensor([[0.2937, 0.6310, 0.0921, 0.4337],

[0.4309, 0.4937, 0.4258, 0.3123]])

tensor([[1, 2],

[1, 2],

[1, 1]], dtype=torch.int32)

[tensor([3, 0, 2]), tensor([1, 1, 2]), tensor([1, 1])]

tensor([[0.6031],

[0.5451]])

mini-batch: 2

tensor([[0.3428, 0.3041, 0.4170, 0.6813],

[0.8755, 0.5104, 0.6693, 0.5859]])

tensor([[2, 1],

[1, 2],

[1, 1]], dtype=torch.int32)

[tensor([2, 3, 2]), tensor([0, 0, 2]), tensor([1, 1])]

tensor([[0.5568],

[0.1590]])

initial parameters (weights and bias):

[[-0.34693 0.19553]

[-0.18123 0.19197]

[ 0.05438 -0.11105]

[ 0.42513 0.34167]]

[[-0.16466 -0.52702]

[-0.22543 -0.11757]

[ 0.23667 0.57199]]

[[-0.20377 0.3713 ]

[ 0.13177 0.27111]]

[[-0.16825 -0.58044 -0.39152 -0.64812]

[ 1.11561 0.0879 0.61481 -0.67743]

[ 0.09677 0.62959 -0.17907 0.55115]]

[-0.62618 -0.7872 0.21905]

[[-0.23981 0.40607 -1.25093]

[ 0.45048 1.64331 -0.01557]]

[0.02414 0.12696]

[[-0.76015 0.17397 -0.65541 -0.1746 0.5074 -0.30015 0.20463 0.41345]

[ 0.1138 -0.55969 -0.13573 0.79993 -0.82672 -0.11259 -0.2254 0.04929]

[ 0.30546 0.65675 -0.11032 0.33164 0.20402 0.19365 -0.23022 -0.40715]

[-0.44909 -0.30881 0.13133 0.31066 0.13206 -0.22411 0.73728 0.62007]]

[-0.177 -0.41172 0.06511 0.63365]

[[ 0.19212 0.32132 -0.12244 0.26343]

[ 0.89174 -0.13837 0.08274 0.14654]]

[ 0.20062 -0.99836]

[[-1.53246 -0.83254]]

[0.16794]

time/loss/accuracy (if enabled):

Finished training it 1/3 of epoch 0, -1.00 ms/it, loss 0.018993

Finished training it 2/3 of epoch 0, -1.00 ms/it, loss 0.029700

Finished training it 3/3 of epoch 0, -1.00 ms/it, loss 0.049903

updated parameters (weights and bias):

emb\_l.0.weight

[[-0.34698 0.19556]

[-0.18126 0.19205]

[ 0.05439 -0.11103]

[ 0.4251 0.34159]]

emb\_l.1.weight

[[-0.16464 -0.52695]

[-0.2255 -0.11758]

[ 0.23668 0.57204]]

emb\_l.2.weight

[[-0.20388 0.37132]

[ 0.13175 0.27098]]

bot\_l.0.weight

[[-0.16825 -0.58044 -0.39152 -0.64812]

[ 1.11518 0.08764 0.61439 -0.67776]

[ 0.09677 0.62959 -0.17907 0.55116]]

bot\_l.0.bias

[-0.62618 -0.78773 0.21905]

bot\_l.2.weight

[[-0.23981 0.40607 -1.25093]

[ 0.45048 1.64324 -0.01563]]

bot\_l.2.bias

[0.02414 0.1269 ]

top\_l.0.weight

[[-0.76015 0.17397 -0.65541 -0.1746 0.5074 -0.30015 0.20463 0.41345]

[ 0.1138 -0.55969 -0.13573 0.79993 -0.82672 -0.11259 -0.2254 0.04929]

[ 0.30546 0.65666 -0.11032 0.33164 0.20399 0.19363 -0.23022 -0.40716]

[-0.44909 -0.30864 0.13133 0.31065 0.13211 -0.22405 0.73728 0.62009]]

top\_l.0.bias

[-0.177 -0.41172 0.06508 0.63371]

top\_l.2.weight

[[ 0.19212 0.32132 -0.12192 0.26336]

[ 0.89174 -0.13837 0.08274 0.14654]]

top\_l.2.bias

[ 0.20084 -0.99836]

top\_l.4.weight

[[-1.53243 -0.83254]]

top\_l.4.bias

[0.1678]

Unable to import onnx. No module named 'onnx'

Using CPU...

model arch:

mlp top arch 3 layers, with input to output dimensions:

[3 4 2 1]

# of interactions

3

mlp bot arch 2 layers, with input to output dimensions:

[4 3 2]

# of features (sparse and dense)

2

dense feature size

4

sparse feature size

2

# of embeddings (= # of sparse features) 1, with dimensions 2x:

[6]

data (inputs and targets):

mini-batch: 0

tensor([[0.6965, 0.2861, 0.2269, 0.5513],

[0.7195, 0.4231, 0.9808, 0.6848]])

tensor([[2, 3]], dtype=torch.int32)

[tensor([2, 4, 0, 2, 4])]

tensor([[0.1825],

[0.1755]])

initial parameters (weights and bias):

[[ 0.02576 0.02599]

[ 0.10974 0.28531]

[ 0.18327 0.09065]

[ 0.18162 -0.14455]

[-0.11285 -0.22187]

[-0.16843 0.10694]]

[[-0.07487 -0.46063 -0.13663 -1.49591]

[-0.94692 -0.3741 0.49575 -0.09281]

[ 0.00152 0.36787 -0.47013 0.15161]]

[-0.46498 -0.99747 -0.22569]

[[ 0.36291 0.21414 -0.00748]

[ 1.51306 0.26115 0.61901]]

[ 1.58261 -0.91506]

[[-0.55526 0.93205 -0.42658]

[ 0.01587 0.57157 0.4761 ]

[ 0.93803 0.79946 0.57161]

[-0.41303 0.42487 0.16799]]

[-0.66313 0.70865 0.40362 0.02275]

[[-0.13458 -0.69184 0.1152 0.27045]

[-0.47987 0.671 -0.63347 -1.22577]]

[ 0.7352 -0.28522]

[[-0.1029 -0.68383]]

[-1.60596]

time/loss/accuracy (if enabled):

Finished training it 1/1 of epoch 0, -1.00 ms/it, loss 0.000323

updated parameters (weights and bias):

emb\_l.0.weight

[[ 0.02576 0.02599]

[ 0.10974 0.28531]

[ 0.18327 0.09065]

[ 0.18162 -0.14455]

[-0.11285 -0.22187]

[-0.16843 0.10694]]

bot\_l.0.weight

[[-0.07487 -0.46063 -0.13663 -1.49591]

[-0.94692 -0.3741 0.49575 -0.09281]

[ 0.00152 0.36787 -0.47013 0.15161]]

bot\_l.0.bias

[-0.46498 -0.99747 -0.22569]

bot\_l.2.weight

[[ 0.36291 0.21414 -0.00748]

[ 1.51306 0.26115 0.61901]]

bot\_l.2.bias

[ 1.58261 -0.91506]

top\_l.0.weight

[[-0.55526 0.93205 -0.42658]

[ 0.01587 0.57157 0.4761 ]

[ 0.93803 0.79946 0.57161]

[-0.41303 0.42487 0.16799]]

top\_l.0.bias

[-0.66313 0.70865 0.40362 0.02275]

top\_l.2.weight

[[-0.13458 -0.69184 0.11519 0.27045]

[-0.47987 0.671 -0.63347 -1.22577]]

top\_l.2.bias

[ 0.73519 -0.28522]

top\_l.4.weight

[[-0.10288 -0.68383]]

top\_l.4.bias

[-1.60592]

initial parameters (weights and bias):

emb\_l.0.weight

[[ 0.02576 0.02599]

[ 0.10974 0.28531]

[ 0.18327 0.09065]

[ 0.18162 -0.14455]

[-0.11285 -0.22187]

[-0.16843 0.10694]]

ly [tensor([[ 0.0704, -0.1312],

[ 0.0962, -0.1052]], grad\_fn=<EmbeddingBagBackward0>)]

\*offset\* \*lS\_o\* tensor([[0, 2]])

\*input\* \*lS\_i\* [tensor([2, 4, 0, 2, 4])]

0.18327 0.09065+

-0.11285 -0.22187

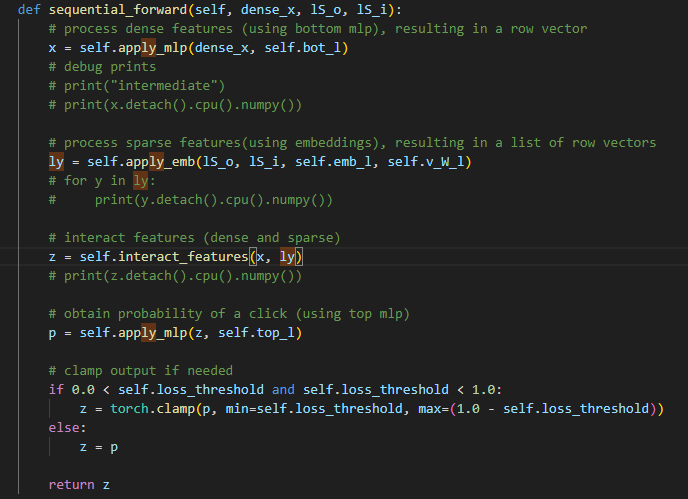
=0.07042, -0.13122

024=>

0.07042, -0.13122+

0.02576, 0.02599

0.09618, -0.10523



X = bot mlp out put

Ly = embedding gather reduce output

Z = interact ouput

P = top mlp out put

d:; cd 'd:\RA data\dlrm-main'; & 'c:\Users\White\AppData\Local\Programs\Python\Python39\python.exe' 'D:\RA data\dlrm-main\dlrm\_s\_pytorch.py' --mini-batch-size=2 --data-size=2 --debug-mode --numpy-rand-seed=123 --arch-sparse-feature-size=64 --arch-mlp-bot=256-128-64 --arch-mlp-top=512-128-1 --num-indices-per-lookup=40 --arch-embedding-size=10000000

d:; cd 'd:\RA data\dlrm-main'; & 'c:\Users\White\AppData\Local\Programs\Python\Python39\python.exe' 'D:\RA data\dlrm-main\dlrm\_s\_pytorch.py' --mini-batch-size=2 --data-size=2 --debug-mode --numpy-rand-seed=123 --arch-sparse-feature-size=64 --arch-mlp-bot=256-128-64 --arch-mlp-top=512-128-1 --num-indices-per-lookup=40 --arch-embedding-size=10000000

python3 dlrm\_s\_pytorch.py --mini-batch-size=2 --data-size=2 --debug-mode --numpy-rand-seed=123 --arch-sparse-feature-size=64 --arch-mlp-bot=256-128-64 --arch-mlp-top=512-128-1 --num-indices-per-lookup=40 --arch-embedding-size=10000000

python3 dlrm\_s\_pytorch.py --mini-batch-size=2048 --data-size=2048 --numpy-rand-seed=123 --arch-sparse-feature-size=64 --arch-mlp-bot=256-128-64 --arch-mlp-top=512-128-1 --num-indices-per-lookup=40 --arch-embedding-size=10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000-10000000

d:; cd 'd:\RA data\dlrm-main'; & 'c:\Users\White\AppData\Local\Programs\Python\Python39\python.exe' 'D:\RA data\dlrm-main\dlrm\_s\_pytorch.py' --mini-batch-size=2 --data-size=2 --numpy-rand-seed=123 --arch-embedding-size=6 --enable-profiling

d:; cd 'd:\RA data\dlrm-main'; & 'c:\Users\White\AppData\Local\Programs\Python\Python39\python.exe' 'D:\RA data\dlrm-main\dlrm\_s\_pytorch.py' --arch-embedding-size=1000000-1000000-1000000-1000000-1000000-1000000-1000000-1000000 --arch-sparse-feature-size=64 --arch-mlp-bot=512-512-64 --arch-mlp-top=1024-1024-1024-1 --data-generation=random --mini-batch-size=2048 --num-batches=1000 --num-indices-per-lookup=100

python3 dlrm\_s\_pytorch.py --arch-embedding-size=1000000-1000000-1000000-1000000-1000000-1000000-1000000-1000000 --arch-sparse-feature-size=64 --arch-mlp-bot=512-512-64 --arch-mlp-top=1024-1024-1024-1 --data-generation=random --mini-batch-size=2048 --num-batches=1000 --num-indices-per-lookup=100 --enable-profiling

d:; cd 'd:\RA data\dlrm-main'; & 'c:\Users\White\AppData\Local\Programs\Python\Python39\python.exe' 'D:\RA data\dlrm-main\dlrm\_s\_pytorch.py' --mini-batch-size=2 --data-size=2 --numpy-rand-seed=123 --arch-embedding-size=6

d:; cd 'd:\RA data\dlrm-main'; & 'c:\Users\White\AppData\Local\Programs\Python\Python39\python.exe' 'D:\RA data\dlrm-main\dlrm\_s\_pytorch.py' --arch-embedding-size=1000000-1000000-1000000-1000000-1000000-1000000-1000000-1000000 --arch-sparse-feature-size=64 --arch-mlp-bot=512-512-64 --arch-mlp-top=1024-1024-1024-1 --data-generation=random --mini-batch-size=2048 --num-batches=1 --num-indices-per-lookup=100 --enable-profiling --nepochs=10

d:; cd 'd:\RA data\dlrm-main'; & 'c:\Users\White\AppData\Local\Programs\Python\Python39\python.exe' 'D:\RA data\dlrm-main\dlrm\_s\_pytorch.py' --mini-batch-size=2 --data-size=2 --numpy-rand-seed=123 --arch-embedding-size=6-6 --debug-mode

d:; cd 'd:\RA data\dlrm-main'; & 'c:\Users\White\AppData\Local\Programs\Python\Python39\python.exe' 'D:\RA data\dlrm-main\dlrm\_s\_pytorch.py' --mini-batch-size=2 --data-size=2 --numpy-rand-seed=123 --arch-embedding-size=6-6 --debug-mode