به نام خداوند جان و خرد

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

سامانه‌های یادگیری‌ماشین توزیع شده

تمرین کامپیوتری شماره ۴

نام و نام خانوادگی: علی خرم فر

شماره دانشجویی: ۸۱۰۱۰۲۱۲۹

دی‌ماه ۱۴۰۳

فهرست مطالب

فهرست جداول

فهرست اشکال

۱_پاسخ سوال شماره ۱

۱-۱_ طراحی مدل

برای شروع این تمرین، اولین گام طراحی مدلی ساده بود که بتوانیم آن را در محیط توزیع‌شده آموزش دهیم و به دقت بالای ۸۰ برسیم. با توجه به جزئیات خواسته شده در متن سوال، مدلی شامل لایه‌های FeedForward، ReLU و BatchNorm طراحی شد.

```
class SimpleModel(nn.Module):
  def __init__(self, input_size, hidden_size, output_size):
    super(SimpleModel, self).__init()__
    self.layer1 = nn.Linear(input_size, hidden_size)
    self.bn1 = nn.BatchNorm1d(hidden_size)
    self.relu = nn.ReLU()
    self.layer2 = nn.Linear(hidden_size, output_size)

  def forward(self, x):
    x = self.layer1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.layer2(x)
    return x
```

لایه Linear: این لایه برای تبدیل ویژگی‌های ورودی به فضای جدید به کار می‌رود و به ما اجازه می‌دهد ویژگی‌های مهم‌تری را از داده‌ها استخراج کنیم. این ویژگی‌ها به مدل کمک میکنند تا بهتر الگوهای موجود را شناسایی کند.

BatchNorm1d: برای جلوگیری از مشکلاتی مانند تغییر توزیع داده‌ها در طول فرآیند آموزش، از نرمال‌سازی دسته‌ای استفاده کردیم.

ReLU: این تابع فعالساز یکی از رایج‌ترین توابع در طراحی شبکه‌های عصبی است که اعداد بالاتر از صفر را حفظ کرده و کمتر برابر صفر می‌شود.

لایه خروجی Linear: این لایه خروجی مدل را به اندازه کلاس‌های داده کاهش میدهد.

۱-۲_ آماده‌سازی داده‌ها

ابتدا، داده‌ها را از فایل‌های ذخیره‌شده با فرمت NumPy بارگذاری کردیم. داده‌های آموزشی و تست شامل ویژگی‌ها و برچسب‌های آن‌ها بودند. باتوجه به توضیحات ارائه‌شده در فایل تمرین، داده‌ها را به نوع float32 تبدیل شدند.

کد مربوط به بارگذاری داده‌ها به صورت زیر است:

```
def load_data():
  train_x = np.load('train_data/train_x.npy').astype(np.float32)
  train_y = np.load('train_data/train_y.npy')
  test_x = np.load('test_data/test_x.npy').astype(np.float32)
  test_y = np.load('test_data/test_y.npy')

  return train_x, train_y, test_x, test_y
```

۳-۱_ تابع آموزش

این تابع بخش‌های مختلفی دارد که به بررسی آن‌ها خواهیم پرداخت. در این سوال از کتابخانه PyTorch DistributedDataParallel (DDP) استفاده کردیم.

تنظیم محیط و آماده‌سازی داده‌ها

```
  dist.init_process_group("gloo", rank=rank, world_size=world_size)
```

از dist.init_process_group برای آغاز ارتباط بین فرآیندهای مختلف استفاده شد. این دستور به طور خودکار ارتباطات لازم را در بستر gloo برقرار می‌کند. مقادیر رنک و worldsize در قسمت main فایل پایتون به صورت زیر دریافت شد:

```
if __name__ == "__main__":
  world_size = int(os.environ.get("WORLD_SIZE", 1))
  rank = int(os.environ.get("RANK", 0))
  train(rank, world_size)
```

تابع معرفی شده در قسمت قبلی برای بارگذاری داده فراخواهی شد و تعداد کلاس‌ها با استفاده از توابع NumPy از داده‌های آموزشی استخراج شد.

```
  train_x, train_y, test_x, test_y = load_data()
  num_classes = len(np.unique(train_y))
  train_x = torch.tensor(train_x)
  train_y = torch.tensor(train_y)
  test_x = torch.tensor(test_x)
  test_y = torch.tensor(test_y)
```

ویژگی‌ها به float32 تبدیل و سپس به Tensors تبدیل شدند تا در PyTorch قابل استفاده باشند.

برای توزیع داده‌ها بین فرآیندها، از DistributedSampler استفاده کردیم. این ابزار به ما کمک می‌کند که هر فرآیند به داده‌های مخصوص به خود دسترسی داشته باشد. با استفاده از DataLoader، داده‌ها در دسته‌های batch تقسیم شدند تا بتوان آن‌ها را در طول فرآیند آموزش استفاده کرد.

```
  train_dataset = torch.utils.data.TensorDataset(train_x, train_y)
  train_sampler = torch.utils.data.distributed.DistributedSampler(train_dataset, num_replicas=world_size, rank=rank)
  train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32, sampler=train_sampler)

  test_dataset = torch.utils.data.TensorDataset(test_x, test_y)
  test_sampler = torch.utils.data.distributed.DistributedSampler(test_dataset, num_replicas=world_size, rank=rank)
  test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=32, sampler=test_sampler,shuffle=False)
```

تنظیمات device و مدل

مدل طراحی‌شده بخش قبلی به CPU منتقل شدسپس با استفاده از ابزار DistributedDataParallel توزیع شد. باتوجه به DDP گرادیان‌ها به صورت خودکار بین فرآیندها به اشتراک گذاشته خواهندشد.

```
device = torch.device("cpu")
  model = SimpleModel(input_size=512, hidden_size=32, output_size=num_classes).to(device)
  ddp_model = DDP(model)
```

ابعاد ورودی با کمک متد shape از نامپای دریافت شد که ۵۱۲ بود. تعداد کلاس‌های خروجی هم با توجه به خروجی تابع دریافت مقادیر منحصر به فرد ۲۰ بود. ولی خب ما همان خروجی متغیر را اینجا به صورت ورودی دادیم.

برای محاسبه خطای مدل از CrossEntropyLoss استفاده شد بهینه‌سازی مدل نیز با استفاده از الگوریتم Adam انجام می‌شود. نرخ یادگیری هم که در کد مشخص است.

```
  criterion = nn.CrossEntropyLoss()
  optimizer = optim.Adam(ddp_model.parameters(), lr=0.001)
```

۴-۱_ حلقه آموزش

در اولین مرحله پیام آغاز فرآیند آموزش به همراه شماره rank نمایش داده شد تا بتوانیم وضعیت اجرای هر فرآیند را بررسی کنیم.

```
  print(f"Training Started for Rank = {rank}")
```

در هر ایپاک به کمک DistributedSampler داده‌ها به طور تصادفی اما منظم در بین فرآیندها توزیع میشود.

ابتدا و انتهای حلقه نیز از time.time() استفاده شد تا زمان را ثبت کنیم.

```
  start_time = time.time()
  num_epochs = 10
  final_train_accuracy = 0
  final_test_accuracy = 0

  for epoch in range(num_epochs):
#    Training Phase
    ddp_model.train()
    train_sampler.set_epoch(epoch)
    epoch_train_loss = 0
    correct_train = 0
    total_train = 0

    for batch_x, batch_y in train_loader:
      batch_x = batch_x.to(device)
      batch_y = batch_y.to(device)

      optimizer.zero_grad()
      outputs = ddp_model(batch_x)
      loss = criterion(outputs, batch_y)
      loss.backward()
```

```
    optimizer.step()

    epoch_train_loss += loss.item()
    _, predicted = torch.max(outputs, 1)
    total_train += batch_y.size(0)
    correct_train += (predicted == batch_y).sum().item()

  train_loss = epoch_train_loss / len(train_loader)
  train_accuracy = 100 * correct_train / total_train

#   Evaluation Phase
  ddp_model.eval()
  epoch_test_loss = 0
  correct_test = 0
  total_test = 0

  with torch.no_grad():
    for batch_x, batch_y in test_loader:
      batch_x = batch_x.to(device)
      batch_y = batch_y.to(device)

      outputs = ddp_model(batch_x)
      loss = criterion(outputs, batch_y)

      epoch_test_loss += loss.item()
      _, predicted = torch.max(outputs, 1)
      total_test += batch_y.size(0)
      correct_test += (predicted == batch_y).sum().item()

  test_loss = epoch_test_loss / len(test_loader)
  test_accuracy = 100 * correct_test / total_test

  if rank == 0:
    print(f"Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.2f}%, "
      f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.2f}%")

  final_train_accuracy = train_accuracy
  final_test_accuracy = test_accuracy

total_time = time.time() - start_time
```

در rank=0، نتایج هر epoch شامل دقت و خطاهای آموزشی و تست در خروجی نمایش داده شد.

ذخیره‌سازی چک‌پوینت و destroy_process_group

در این بخش از کد فرآیند آموزش به پایان رسیده و نتایج نهایی ثبت شده‌اند. در نهایت باتوجه به قسمت ب سوال مدل آموزش‌دیده به صورت چک‌پوینت ذخیره شد تا بتوان از آن برای استفاده‌های بعدی یا ادامه آموزش بهره برد. باتوجه به کد ارائه شده در یکی از لینک‌های فایل تمرین، از dist.barrier() استفاده شد تا تمام فرایندها منتظر اتمام فعالیت ذخیره بمانند.

```
if rank == 0:
  print(f"Final Train Accuracy: {final_train_accuracy:.2f}%")
  print(f"Final Test Accuracy: {final_test_accuracy:.2f}%")
  print(f"Total Training Time: {total_time:.2f} seconds")

  CHECKPOINT_PATH = "model_checkpoint.pth"
  torch.save(ddp_model.state_dict(), CHECKPOINT_PATH)

dist.barrier()
dist.destroy_process_group()
```

در نهایت process_group توزیع‌شده حذف شد تا منابع سیستم آزاد شوند.

۵-۱_ آموزش به کمک Torchrun و Slurm

یک ماشین و یک هسته

Torchrun ابزاری برای هماهنگی بین فرآیندهای PyTorch در محیط توزیع‌شده به کمک Slurm است.

اسکریپت برای اجرای این حالت به صورت زیر است:

```
/#bin/bash

#SBATCH --job-name=multi_node_pytorch_torchrun
#SBATCH --partition=partition

#SBATCH --mem=1000mb
#SBATCH --nodes=1

#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1

 #####Number of total processes
echo"--------------------"

echo "Nodelist: " $SLURM_JOB_NODELIST
echo "Number of nodes: " $SLURM_JOB_NUM_NODES

echo "Ntasks per node: " $SLURM_NTASKS_PER_NODE

echo "CPUs per task: 1"

echo"--------------------"

 #Master Port
export MASTER_PORT=24442

export RENDEZVOUS_ID=$RANDOM
export WORLD_SIZE=1

 ###Get the first node name as master address.
export MASTER_ADDR=$(scontrol show hostnames "$SLURM_JOB_NODELIST" | head -n 1)

echo "MASTER_ADDR: $MASTER_ADDR:$MASTER_PORT"
echo"--------------------"

 #Just to suppress a warning
export OMP_NUM_THREADS=1

 #Activate virtual environment
source /home/shared_files/pytorch_venv/bin/activate


 ###Torchrun
srun torchrun --nnodes=1 --nproc_per_node=1 --rdzv_id=$RENDEZVOUS_ID --rdzv_backend=c10d --
rdzv_endpoint=$MASTER_ADDR:$MASTER_PORT Q1_SMSC.py
```

تنظیمات Slurm:

--job-name: نام job که در سیستم ثبت می‌شود و با کمک squeue می‌توان مشاهده کرد.

--partition: مشخص‌کننده بخش یا دسته‌ای از ماشین‌ها که در کلاستر ما همانpartition است.

--mem: مقدار حافظه تخصیص داده‌شده

--nodes: تعداد ماشین‌هایی که وظیفه روی آن‌ها اجرا خواهد شد.

--ntasks-per-node: تعداد فرآیندها در هر ماشین در اینجا، یک فرآیند باتوجه به نکته گفته‌شده در ویدئوی hands on

-- cpus-per-task تعداد هسته cpu تخصیص داده شده

MASTER_PORT: شماره پورتی که فرآیند master استفاده می‌کند.

MASTER_ADDR: آدرس ماشین اصلی.

WORLD_SIZE: تعداد کل فرآیندها

از دستور source برای فعال‌سازی محیط مجازی استفاده شده است که شامل پکیج‌های لازم برای اجرای PyTorch در کلاستر است و همان آدرس ارائه‌شده در فایل تمرین استفاده شد.

اجرای Torchrun:

--nnodes=1: تعداد کل ماشین‌های مورد استفاده.

--nproc_per_node=1: تعداد فرآیندها در هر ماشین.

--rdzv_id و --rdzv_backend: تنظیمات لازم برای هماهنگی فرآیندها

Q1_SMSC.py: کد ارائه شده در بخش قبلی که شامل کد آموزش مدل است.

استفاده از یک ماشین و یک هسته ساده‌ترین حالت اجرای توزیع‌شده است در در اصل توزیع شده نیست :) . این تنظیم به ما کمک می‌کند فرآیند را با حداقل پیچیدگی تست کنیم چون در این حالت کل فرآیند آموزش به یک هسته اختصاص داده می‌شود و نیازی به هماهنگی بین چندین فرآیند نیست.

نتیجه اجرا به صورت زیر بود:

```
----------------------
Nodelist:  raspberrypi-dml0
Number of nodes:  1
Ntasks per node:  1
CPUs per task: 1
 ----------------------
MASTER_ADDR: raspberrypi-dml0:24442
----------------------
Training Started for Rank = 0
Epoch [1/10], Train Loss: 1.0383, Train Accuracy: 73.00%, Test Loss: 0.5136, Test Accuracy: 84.84%
```

```
Epoch [2/10], Train Loss: 0.5098, Train Accuracy: 84.08%, Test Loss: 0.4703, Test Accuracy:
85.36%
Epoch [3/10], Train Loss: 0.4634, Train Accuracy: 85.22%, Test Loss: 0.4609, Test Accuracy:
85.49%
Epoch [4/10], Train Loss: 0.4386, Train Accuracy: 86.10%, Test Loss: 0.4547, Test Accuracy:
85.79%
Epoch [5/10], Train Loss: 0.4227, Train Accuracy: 86.35%, Test Loss: 0.4541, Test Accuracy:
85.78%
Epoch [6/10], Train Loss: 0.4129, Train Accuracy: 86.67%, Test Loss: 0.4509, Test Accuracy:
85.78%
Epoch [7/10], Train Loss: 0.4030, Train Accuracy: 87.04%, Test Loss: 0.4530, Test Accuracy:
85.66%
Epoch [8/10], Train Loss: 0.3952, Train Accuracy: 87.28%, Test Loss: 0.4499, Test Accuracy:
85.67%
Epoch [9/10], Train Loss: 0.3879, Train Accuracy: 87.50%, Test Loss: 0.4550, Test Accuracy:
85.73%
Epoch [10/10], Train Loss: 0.3810, Train Accuracy: 87.61%, Test Loss: 0.4545, Test Accuracy:
85.55%
Final Train Accuracy: 87.61%
Final Test Accuracy: 85.55%
Total Training Time: 198.99 seconds
```

در طول ۱۰ ایپاک، مشاهده شد که خطای آموزشی مدل به تدریج کاهش یافته و دقت آن به صورت پیوسته افزایش پیدا کرده است. در پایان، مدل به دقت آموزشی **٪۸۷,۶۱** که نشان‌دهنده پایداری در آموزش مدل و همگرایی است.

از طرف دیگر، دقت تست مدل در پایان آموزش به **٪۸۵,۵۵** رسید و اختلاف جزئی بین دقت آموزش و تست، منطقی و نشان‌دهنده عدم Overfitting است.

آموزش مدل در این حالت **۱۹۸,۹۹ ثانیه** به طول انجامید. این زمان با توجه به محدودیت منابع یک ماشین و یک هسته کاملاً منطقی است.

یک ماشین و دو هسته

در اینجا باتوجه به تکراری بودن کدها فقط اشاره می‌کنیم که cpus-per-task- برابر ۲ nproc_per_node در torchrun نیز برابر ۲ قرار داده شد.

```
------------------
Nodelist:  raspberrypi-dml0
Number of nodes:  1
Ntasks per node:  1
CPUs per task: 2
------------------
MASTER_ADDR: raspberrypi-dml0:24442
------------------
Training Started for Rank = 0
Training Started for Rank = 1
Epoch [1/10], Train Loss: 1.2284, Train Accuracy: 69.96%, Test Loss: 0.5491, Test Accuracy:
84.44%
Epoch [2/10], Train Loss: 0.5283, Train Accuracy: 83.84%, Test Loss: 0.4778, Test Accuracy:
85.40%
Epoch [3/10], Train Loss: 0.4567, Train Accuracy: 85.82%, Test Loss: 0.4702, Test Accuracy:
85.26%
```

```
Epoch [4/10], Train Loss: 0.4350, Train Accuracy: 86.21%, Test Loss: 0.4566, Test Accuracy:
85.46%
Epoch [5/10], Train Loss: 0.4197, Train Accuracy: 86.52%, Test Loss: 0.4599, Test Accuracy:
85.54%
Epoch [6/10], Train Loss: 0.4050, Train Accuracy: 87.04%, Test Loss: 0.4570, Test Accuracy:
85.74%
Epoch [7/10], Train Loss: 0.4043, Train Accuracy: 87.00%, Test Loss: 0.4605, Test Accuracy:
85.86%
Epoch [8/10], Train Loss: 0.3886, Train Accuracy: 87.24%, Test Loss: 0.4607, Test Accuracy:
85.34%
Epoch [9/10], Train Loss: 0.3899, Train Accuracy: 87.25%, Test Loss: 0.4547, Test Accuracy:
85.82%
Epoch [10/10], Train Loss: 0.3841, Train Accuracy: 87.53%, Test Loss: 0.4541, Test Accuracy:
85.66%
Final Train Accuracy: 87.53%
Final Test Accuracy: 85.66%
Total Training Time: 165.63 seconds
```

در حالت یک ماشین و دو هسته، مدل توانست به دقت آموزشی **٪۸۷,۵۳** و دقت تست **٪۸۵,۶۶** دست یابد. زمان کل آموزش در این حالت **۱۶۵,۶۳ ثانیه** بود که نسبت به حالت یک هسته کاهش زیادی داشت. استفاده از دو هسته باعث افزایش کارایی و کاهش زمان آموزش شد، دقت نهایی هم بهبود جزئی داشت.

دو ماشین و یک هسته

در اینجا باتوجه به تکراری بودن کدها فقط اشاره می‌کنیم که –nodes برابر ۲ –cpus-per-task برابر ۱ در torchrun نیز nproc_per_node برابر ۱ و –nnodes= برابر ۲ قرار داده شد.

```
----------------------
Nodelist:  raspberrypi-dml[۰-۱]
Number of nodes:  2
Ntasks per node:  1
CPUs per task: 1
 ----------------------
MASTER_ADDR: raspberrypi-dml0:24442
----------------------
Training Started for Rank = 0
Training Started for Rank = 1
Epoch [1/10], Train Loss: 1.2284, Train Accuracy: 69.96%, Test Loss: 0.5491, Test Accuracy:
84.44%
Epoch [2/10], Train Loss: 0.5283, Train Accuracy: 83.84%, Test Loss: 0.4778, Test Accuracy:
85.40%
Epoch [3/10], Train Loss: 0.4567, Train Accuracy: 85.82%, Test Loss: 0.4702, Test Accuracy:
85.26%
Epoch [4/10], Train Loss: 0.4350, Train Accuracy: 86.21%, Test Loss: 0.4566, Test Accuracy:
85.46%
Epoch [5/10], Train Loss: 0.4197, Train Accuracy: 86.52%, Test Loss: 0.4599, Test Accuracy:
85.54%
Epoch [6/10], Train Loss: 0.4050, Train Accuracy: 87.04%, Test Loss: 0.4570, Test Accuracy:
85.74%
Epoch [7/10], Train Loss: 0.4043, Train Accuracy: 87.00%, Test Loss: 0.4605, Test Accuracy:
85.86%
Epoch [8/10], Train Loss: 0.3886, Train Accuracy: 87.24%, Test Loss: 0.4607, Test Accuracy:
85.34%
Epoch [9/10], Train Loss: 0.3899, Train Accuracy: 87.25%, Test Loss: 0.4547, Test Accuracy:
85.82%
Epoch [10/10], Train Loss: 0.3841, Train Accuracy: 87.53%, Test Loss: 0.4541, Test Accuracy:
85.66%
Final Train Accuracy: 87.53%
Final Test Accuracy: 85.66%
Total Training Time: 175.67 seconds
```

در حالت دو ماشین و یک هسته، دقت آموزشی مدل به **٪۸۷,۵۳** و دقت تست به **٪۸۵,۶۶** رسید که مشابه حالت قبلی است. زمان کل آموزش در این حالت **۱۷۵,۶۷ ثانیه** بود که نسبت به حالت دو هسته‌ای افزایش نشان داد. این افزایش به دلیل احتمالا به دلیل هماهنگی بیشتر بین ماشین‌ها رخ داده است، اما همچنان عملکرد کلی مدل حفظ شده است.

یک نکته این بود که در اجرای سوال ۲ این حالت زمان کمتری نسبت به ۲ هسته روی یک ماشین داشت که عجیب بود باتوجه به نبود زمان برای آموزش بیشتر این احتمال وجود دارد که در این اجرا زمان بیشتر ثبت شده است. ولی خب ممکن است تفاوت ساختار accelerate و torchrunهم علت این موضوع باشد.

دو ماشین و دو هسته

در اینجا باتوجه به تکراری بودن کدها فقط اشاره می‌کنیم که –nodes برابر ۲ –cpus-per-task برابر ۲ در torchrun نیز nproc_per_node برابر ۲ و –-nnodes= برابر ۲ قرار داده شد.

```
----------------------
Nodelist:  raspberrypi-dml[۰-۱]
Number of nodes:  2
Ntasks per node:  1
CPUs per task: 2
 ----------------------
MASTER_ADDR: raspberrypi-dml0:24442
----------------------
Training Started for Rank = 1
Training Started for Rank = 0
Training Started for Rank = 3Training Started for Rank = 2

Epoch [1/10], Train Loss: 1.5180, Train Accuracy: 65.01%, Test Loss: 0.6672, Test Accuracy:
84.52%
Epoch [2/10], Train Loss: 0.5846, Train Accuracy: 83.86%, Test Loss: 0.4952, Test Accuracy:
86.08%
Epoch [3/10], Train Loss: 0.4744, Train Accuracy: 85.38%, Test Loss: 0.4616, Test Accuracy:
86.08%
Epoch [4/10], Train Loss: 0.4432, Train Accuracy: 86.13%, Test Loss: 0.4472, Test Accuracy:
86.40%
Epoch [5/10], Train Loss: 0.4219, Train Accuracy: 86.65%, Test Loss: 0.4522, Test Accuracy:
85.80%
Epoch [6/10], Train Loss: 0.4066, Train Accuracy: 86.85%, Test Loss: 0.4612, Test Accuracy:
86.48%
Epoch [7/10], Train Loss: 0.4049, Train Accuracy: 87.33%, Test Loss: 0.4616, Test Accuracy:
86.20%
Epoch [8/10], Train Loss: 0.3802, Train Accuracy: 87.71%, Test Loss: 0.4593, Test Accuracy:
85.84%
Epoch [9/10], Train Loss: 0.3919, Train Accuracy: 87.50%, Test Loss: 0.4653, Test Accuracy:
85.76%
Epoch [10/10], Train Loss: 0.3754, Train Accuracy: 87.87%, Test Loss: 0.4589, Test Accuracy:
85.92%
Final Train Accuracy: 87.87%
Final Test Accuracy: 85.92%
Total Training Time: 107.69 seconds
```

در حالت دو ماشین و دو هسته، مدل به دقت آموزشی **٪۸۷,۸۷** و دقت تست **٪۸۵,۹۲** دست یافت که نشان‌دهنده بهبود جزئی در دقت تست نسبت به حالت‌های قبلی است. زمان کل آموزش به **۱۰۷,۶۹ ثانیه** کاهش یافت، که سریع‌ترین زمان در میان تمام حالت‌ها بود. این کاهش به دلیل بهره‌برداری موثر از دو هسته در هر ماشین و تقسیم بهتر بار محاسباتی بین ماشین‌ها حاصل شد.

۶-۱_ تحلیل نتایج

کمترین زمان آموزش مربوط به حالت دو ماشین و دو هسته است که به دلیل استفاده همزمان از دو ماشین و چندین هسته حاصل شد. زمان آموزش در حالت یک ماشین و یک هسته بیشترین مقدار را داشت، که نشان‌دهنده کندی ناشی از محدودیت منابع است. دقت آموزشی و تست در تمام حالت‌ها نسبتا مشابه بود، که نشان‌دهنده پایداری مدل در شرایط مختلف است و می‌توانیم اطمینان حاصل کنیم که وقتی یادگیری ما توزیع شده است دقت مدل کاهش نخواهدیافت. این آزمایش نشان داد که افزایش تعداد هسته‌ها و ماشین‌ها می‌تواند زمان آموزش را تا حد زیادی کاهش دهد.

حالت	دقت آموزش	دقت تست	زمان آموزش (ثانیه)
۱ ماشین، ۱ هسته	۸۷,۶۱	۸۵,۵۵	۱۹۸,۹۹
۱ ماشین، ۲ هسته	۸۷,۵۳	۸۵,۶۶	۱۶۵,۶۳
۲ ماشین، ۱ هسته برای هر ماشین	۸۷,۵۳	۸۵,۶۶	۱۷۵,۶۷
۲ ماشین، ۲ هسته برای هر ماشین	۸۷,۸۷	۸۵,۹۲	۱۰۷,۶۹

جدول ۱ نتایج آزمایش با کمک Torchrun

۲_ پاسخ سوال شماره ۲

هدفمان در این قسمت استفاده از کتابخانه Huggingface Accelerate برای ساده‌سازی فرآیند آموزش مدل به صورت توزیع‌شده است.

۲-۱_ تغییرات در پیاده‌سازی و آموزش مدل

در فایل‌های پیوست شده تغییراتی که اعمال شده اند به صورت کامنت مشخص شده‌اند.

ابزارهای توزیع‌شده PyTorch مانند DDP و DistributedSampler دیگر نیازی به استفاده ندارند، زیرا کتابخانه Accelerate مدیریت فرآیند توزیع و هماهنگی را بر عهده می‌گیرد.

اضافه کردن Accelerator:

```
accelerator = Accelerator()
```

یک شیء از کلاس Accelerator ایجاد شد که مسئول مدیریت فرآیندهای توزیع‌شده است. این شیء توابعی برای آماده‌سازی مدل، داده‌ها و بهینه‌ساز فراهم می‌کند. نمونه‌گیری داده‌ها به صورت تصادفی انجام شد و نیازی به DistributedSampler نبود.

استفاده از accelerator.prepare:

```
 model, optimizer, train_loader, test_loader = accelerator.prepare(
   model, optimizer, train_loader, test_loader
# ( Added for accelerate
```

مدل، بهینه‌ساز و DataLoader با استفاده از accelerator.prepare برای استفاده در حالت توزیع‌شده آماده شدند. این مرحله جایگزین تنظیمات پیچیده توزیع PyTorch شد.

مدیریت گرادیان‌ها:

```
    accelerator.backward(loss)  # Added for accelerate
```

در فرآیند آموزش، به جای استفاده از loss.backward از accelerator.backward استفاده شد که گرادیان‌ها بین فرآیندها به اشتراک گذاشته شوند.

مدیریت خروجی‌ها:

```
 if accelerator.is_main_process:
   print(accelerator.distributed_type)
```

برای اطمینان از جلوگیری از تداخل بین فرآیندها، از accelerator.is_main_process برای چاپ خروجی تنها توسط فرآیند اصلی استفاده شد.

```
 print(f"Training Started for Rank = {accelerator.process_index}")
```

همچنین شماره رنک هر فرایند هم خروجی گرفته شد.

حذف نیاز به dist.init_process_group و dist.destroy_process_group:

فرآیندهای هماهنگی بین هسته‌ها و ماشین‌ها به طور خودکار توسط Accelerate مدیریت می‌شوند، بنابراین نیازی به دستورات مربوطه نبود.

ذخیره‌سازی مدل:

ذخیره مدل تنها توسط فرآیند اصلی انجام شد که توسط شرط accelerator.is_main_process کنترل شد.

۲-۲_ تغییرات در اسکریپت Slurm

در این اسکریپت، تغییرات و تنظیمات جدیدی اعمال شد. نام job را عوض کردیم و همچنین مقدار حافظه مورد استفاده به ۱۵۰۰ مگابایت افزایش یافت تا از خطاهای ناشی از کمبود حافظه جلوگیری شود.

دستور accelerate launch به جای torchrun استفاده شد تا فرآیندهای توزیع‌شده با Accelerate اجرا شوند.

آرگومان‌های مرتبط با Accelerate شامل موارد زیر بودند:

--num_processes: تعداد کل فرآیندها.

--num_machines: تعداد ماشین‌ها

--rdzv_backend: نوع ارتباط فرآیندها c10d

--main_process_ip و --main_process_port: آدرس و پورت ماشین اصلی.

--dynamo_backend=no: غیرفعال‌سازی بهینه‌سازی Dynamo.

--mixed_precision=no: غیرفعال‌سازی Mixed Precision.

--machine_rank: رنک ماشین مشخص‌شده توسط Slurm.

--multi_gpu: فعال‌سازی قابلیت چند GPU (و یا چند CPU😁).

هرچند برای بخش اول که یک نود و یک هسته بود multi_gpu را قرار ندادیم.

یک ماشین و یک هسته

باتوجه به تکرار بودن کد از ارائه مجدد توضیحات صرف نظر شد. فقط مقدار ماشین‌ها در اسکریپت اجرا به ۱ ماشین و یک هسته تغییر یافت.

```
----------------------
Nodelist:  raspberrypi-dml0
Number of nodes:  1
Ntasks per node:  1
CPUs per task: 1
 ----------------------
MASTER_ADDR: raspberrypi-dml0:24442
----------------------
DistributedType.NO
Training Started for Rank = 0
Epoch [1/10], Train Loss: 1.0441, Train Accuracy: 72.87%, Test Loss: 0.5069, Test Accuracy:
84.74%
Epoch [2/10], Train Loss: 0.5082, Train Accuracy: 84.11%, Test Loss: 0.4651, Test Accuracy:
85.23%
Epoch [3/10], Train Loss: 0.4657, Train Accuracy: 85.14%, Test Loss: 0.4537, Test Accuracy:
85.60%
Epoch [4/10], Train Loss: 0.4372, Train Accuracy: 85.99%, Test Loss: 0.4507, Test Accuracy:
85.69%
Epoch [5/10], Train Loss: 0.4248, Train Accuracy: 86.30%, Test Loss: 0.4509, Test Accuracy:
85.94%
Epoch [6/10], Train Loss: 0.4127, Train Accuracy: 86.83%, Test Loss: 0.4536, Test Accuracy:
85.39%
Epoch [7/10], Train Loss: 0.4012, Train Accuracy: 86.99%, Test Loss: 0.4516, Test Accuracy:
85.82%
Epoch [8/10], Train Loss: 0.3947, Train Accuracy: 87.28%, Test Loss: 0.4570, Test Accuracy:
85.65%
Epoch [9/10], Train Loss: 0.3865, Train Accuracy: 87.42%, Test Loss: 0.4493, Test Accuracy:
86.06%
```

```
Epoch [10/10], Train Loss: 0.3832, Train Accuracy: 87.56%, Test Loss: 0.4553, Test Accuracy:
85.67%
Final Train Accuracy: 87.56%
Final Test Accuracy: 85.67%
Total Training Time: 191.17 seconds
```

مدل در این حالت توانست به دقت آموزشی **٪۸۷,۵۶** برسد، که نشان‌دهنده آموزش خوب مدل با داده‌های آموزشی است. دقت تست **٪۸۵,۶۷** هم نشان می‌دهد که مدل تعمیم کافی روی داده‌های تست دارد. مدت زمان صرف‌شده برای آموزش مدل در این حالت **۱۹۱,۱۷ ثانیه** بود.

یک ماشین و دو هسته

باتوجه به تکرار بودن کد از ارائه مجدد توضیحات صرف نظر شد. فقط مقدار ماشین‌ها در اسکریپت اجرا به ۱ ماشین و ۲ هسته تغییر یافت.

```
----------------------
Nodelist:  raspberrypi-dml0
Number of nodes:  1
Ntasks per node:  1
CPUs per task: 2
 ----------------------
MASTER_ADDR: raspberrypi-dml0:24442
----------------------
W0117 02:39:57.059000 1604288 torch/distributed/run.py:785] master_addr is only used for static
rdzv_backend and when rdzv_endpoint is not specified.
DistributedType.MULTI_CPU
Training Started for Rank = 0Training Started for Rank = 1

Epoch [1/10], Train Loss: 1.2321, Train Accuracy: 70.07%, Test Loss: 0.5478, Test Accuracy:
84.22%
Epoch [2/10], Train Loss: 0.5222, Train Accuracy: 84.35%, Test Loss: 0.4639, Test Accuracy:
84.97%
Epoch [3/10], Train Loss: 0.4628, Train Accuracy: 85.54%, Test Loss: 0.4601, Test Accuracy:
85.47%
Epoch [4/10], Train Loss: 0.4344, Train Accuracy: 86.17%, Test Loss: 0.4577, Test Accuracy:
85.17%
Epoch [5/10], Train Loss: 0.4168, Train Accuracy: 86.69%, Test Loss: 0.4561, Test Accuracy:
85.63%
Epoch [6/10], Train Loss: 0.4068, Train Accuracy: 87.00%, Test Loss: 0.4523, Test Accuracy:
85.43%
Epoch [7/10], Train Loss: 0.4036, Train Accuracy: 86.95%, Test Loss: 0.4498, Test Accuracy:
85.35%
Epoch [8/10], Train Loss: 0.3951, Train Accuracy: 87.38%, Test Loss: 0.4578, Test Accuracy:
85.43%
Epoch [9/10], Train Loss: 0.3903, Train Accuracy: 87.18%, Test Loss: 0.4546, Test Accuracy:
85.47%
Epoch [10/10], Train Loss: 0.3747, Train Accuracy: 87.68%, Test Loss: 0.4540, Test Accuracy:
85.51%
Final Train Accuracy: 87.68%
Final Test Accuracy: 85.51%
Total Training Time: 176.73 seconds
```

مدل در این حالت به دقت آموزشی **٪۸۷,۶۸** رسید، که نشان‌دهنده توانایی مدل در یادگیری دقیق‌تر با توزیع بهتر داده‌ها در دو هسته است. همچنین، دقت تست **٪۸۵,۵۱** شد. زمان کل آموزش در این حالت **۱۷۶,۷۳ ثانیه** بود که نسبت به حالت تک‌هسته‌ای کاهش داشت.

یک ماشین و یک هسته

باتوجه به تکرار بودن کد از ارائه مجدد توضیحات صرف نظر شد. فقط مقدار ماشین‌ها در اسکریپت اجرا به ۱ ماشین و ۱ هسته تغییر یافت. البته در اینجا باید تعداد فرایندها را در قسمت Accelerate۲ بگذاریم و این تفاوتش با torchrun است که تعداد کل مد نظر است.

```
----------------------
Nodelist:  raspberrypi-dml[۰-۱]
Number of nodes:  2
Ntasks per node:  1
CPUs per task: 1
 ----------------------
MASTER_ADDR: raspberrypi-dml0:24442
----------------------
DistributedType.MULTI_CPU
Training Started for Rank = 0
Training Started for Rank = 1
Epoch [1/10], Train Loss: 1.2321, Train Accuracy: 70.07%, Test Loss: 0.5478, Test Accuracy:
84.22%
Epoch [2/10], Train Loss: 0.5222, Train Accuracy: 84.35%, Test Loss: 0.4639, Test Accuracy:
84.97%
Epoch [3/10], Train Loss: 0.4628, Train Accuracy: 85.54%, Test Loss: 0.4601, Test Accuracy:
85.47%
Epoch [4/10], Train Loss: 0.4344, Train Accuracy: 86.17%, Test Loss: 0.4577, Test Accuracy:
85.17%
Epoch [5/10], Train Loss: 0.4168, Train Accuracy: 86.69%, Test Loss: 0.4561, Test Accuracy:
85.63%
Epoch [6/10], Train Loss: 0.4068, Train Accuracy: 87.00%, Test Loss: 0.4523, Test Accuracy:
85.43%
Epoch [7/10], Train Loss: 0.4036, Train Accuracy: 86.95%, Test Loss: 0.4498, Test Accuracy:
85.35%
Epoch [8/10], Train Loss: 0.3951, Train Accuracy: 87.38%, Test Loss: 0.4578, Test Accuracy:
85.43%
Epoch [9/10], Train Loss: 0.3903, Train Accuracy: 87.18%, Test Loss: 0.4546, Test Accuracy:
85.47%
Epoch [10/10], Train Loss: 0.3747, Train Accuracy: 87.68%, Test Loss: 0.4540, Test Accuracy:
85.51%
Final Train Accuracy: 87.68%
Final Test Accuracy: 85.51%
Total Training Time: 133.44 seconds
```

در این تنظیم، مدل به دقت آموزشی **٪۸۷,۶۸** رسید که نشان‌دهنده یادگیری خوب در این حالت توزیع‌شده است. دقت تست **٪۸۵,۵۱** شد. زمان آموزش در این حالت **۱۳۳,۴۴ ثانیه** بود که کاهش قابل‌توجهی نسبت به حالت‌های تک ماشینی نشان می‌دهد. این کاهش به دلیل توزیع بهتر بار محاسباتی بین دو ماشین حاصل شده است.

دو ماشین و دو هسته

باتوجه به تکرار بودن کد از ارائه مجدد توضیحات صرف نظر شد. فقط مقدار ماشین‌ها در اسکریپت اجرا به ۱ ماشین و ۱ هسته تغییر یافت. البته در اینجا باید تعداد فرایندها را در قسمت Accelerate ۴ بگذاریم و این تفاوتش با torchrun است که تعداد کل مد نظر است.

```
----------------------
Nodelist:  raspberrypi-dml[۰-۱]
Number of nodes:  2
Ntasks per node:  1
CPUs per task: 2
 ----------------------
MASTER_ADDR: raspberrypi-dml0:24442
----------------------
DistributedType.MULTI_CPU
Training Started for Rank = 0
Training Started for Rank = 2
Training Started for Rank = 1
Training Started for Rank = 3
Epoch [1/10], Train Loss: 1.5223, Train Accuracy: 65.13%, Test Loss: 0.6866, Test Accuracy:
83.58%
Epoch [2/10], Train Loss: 0.5834, Train Accuracy: 84.13%, Test Loss: 0.4905, Test Accuracy:
85.44%
Epoch [3/10], Train Loss: 0.4792, Train Accuracy: 85.55%, Test Loss: 0.4668, Test Accuracy:
85.44%
Epoch [4/10], Train Loss: 0.4310, Train Accuracy: 86.32%, Test Loss: 0.4545, Test Accuracy:
85.52%
Epoch [5/10], Train Loss: 0.4158, Train Accuracy: 86.77%, Test Loss: 0.4580, Test Accuracy:
85.44%
Epoch [6/10], Train Loss: 0.4055, Train Accuracy: 87.31%, Test Loss: 0.4556, Test Accuracy:
85.32%
Epoch [7/10], Train Loss: 0.3973, Train Accuracy: 87.06%, Test Loss: 0.4487, Test Accuracy:
85.48%
Epoch [8/10], Train Loss: 0.3958, Train Accuracy: 87.45%, Test Loss: 0.4563, Test Accuracy:
85.72%
Epoch [9/10], Train Loss: 0.3848, Train Accuracy: 87.42%, Test Loss: 0.4593, Test Accuracy:
85.72%
Epoch [10/10], Train Loss: 0.3703, Train Accuracy: 87.93%, Test Loss: 0.4535, Test Accuracy:
85.80%
Final Train Accuracy: 87.93%
Final Test Accuracy: 85.80%
Total Training Time: 113.55 seconds
```

در این حالت، مدل توانست به دقت آموزشی **٪۸۷,۹۳** برسد، که بهترین عملکرد آموزشی در میان تمامی حالت‌ها بود. همچنین، دقت تست به **٪۸۵,۸۰** رسید. زمان کل آموزش در این تنظیم **۱۱۳,۵۵ ثانیه** بود که کوتاه‌ترین زمان در مقایسه با سایر حالت‌ها محسوب می‌شود.

۳-۲_ تحلیل نتایج

در این آزمایش، مدل در حالت دو ماشین و دو هسته بهترین عملکرد را ارائه داد و به دقت آموزشی ٪۸۷,۹۳ و دقت تست ٪۸۵,۸۰ دست یافت. این حالت همچنین با ۱۱۳,۵۵ ثانیه کمترین زمان آموزش را داشت، که نشان‌دهنده استفاده بهینه از منابع محاسباتی است. در مقابل، حالت یک ماشین و یک هسته با ۱۹۱,۱۷ ثانیه طولانی‌ترین زمان را به خود اختصاص داد. دقت‌های مدل در حالت‌های مختلف تقریباً مشابه بودند، اما کاهش زمان آموزش در حالت‌های چند ماشینی و چند هسته‌ای قابل توجه بود که بار دیگر مزیت یادگیری ماشین توزیع شده را متوجه شدیم.

هر دو ابزار توانستند مدل را با دقت مشابهی آموزش دهند و زمان آموزش را با افزایش منابع کاهش دهند. Accelerate به دلیل سادگی در پیاده‌سازی و مدیریت بهتر منابع برای پروژه‌هایی با مقیاس بزرگ‌تر مناسب‌تر است. از طرف دیگر، Torchrun کنترل بیشتری به کاربر می‌دهد که در شرایط خاص می‌تواند مفید

باشد. با توجه به وابستگی زمان‌ها به شرایط محیطی کلاستر، تحلیل دقیق‌تر نیازمند آزمایش‌های بیشتری است.

حالت	دقت آموزش	دقت تست	زمان آموزش (ثانیه)
۱ ماشین، ۱ هسته	۸۷,۵۶	۸۵,۶۷	۱۹۱,۱۷
۱ ماشین، ۲ هسته	۸۷,۶۸	۸۵,۵۱	۱۷۶,۷۳
۲ ماشین، ۱ هسته برای هر ماشین	۸۷,۶۸	۸۵,۵۱	۱۳۳,۴۴
۲ ماشین، ۲ هسته برای هر ماشین	۸۷,۹۳	۸۵,۸۰	۱۱۳,۵۵

جدول ۲ نتایج آزمایش با کمک Accelerate

اخطار زیر در حالت ۱ نود ۲ فرایند تغییری در زمان آموزش نداشت ولی ممکن است برای راه‌اندازی اولیه وقتی static بذاریم تغییری وجود داشته باشد. هرچند ارتباطی به تمرین نداشت و کد به خوبی اجرا شد.

```
W0117 02:39:57.059000 1604288 torch/distributed/run.py:785] master_addr is only used for static
rdzv_backend and when rdzv_endpoint is not specified.
DistributedType.MULTI_CPU
```

۳_ پاسخ سوال شماره ۳

۳-۱_ روش Mixed Precision و آرگومان‌های مرتبط در Accelerate

Mixed Precision یک روش بهینه‌سازی برای تسریع آموزش مدل‌های یادگیری عمیق است که با استفاده از ترکیب دقت‌های عددی مختلف (مانند FP16 و FP32) انجام می‌شود. این روش باعث کاهش مصرف حافظه و افزایش سرعت ارتباطات می‌شود.

کتابخانه Accelerate امکان استفاده از این روش را با تنظیم آرگومان --mixed_precision فراهم کرده است. این آرگومان می‌تواند مقادیر مختلفی داشته باشد که هرکدام رفتار متفاوتی در مدیریت دقت محاسباتی دارند. این آرگومان می‌تواند مقادیر زیر را داشته باشد:

:no

مقدار پیش‌فرض (Default) که Mixed Precision غیرفعال است و تمامی محاسبات با دقت کامل (FP32) انجام می‌شوند. این حالت بیشترین مصرف حافظه و زمان اجرا را دارد، اما ساده‌ترین حالت از نظر پیاده‌سازی و پشتیبانی سخت‌افزاری است.

fp16یا Float 16:

در این حالت، محاسبات با دقت FP16 انجام می‌شوند که نصف بیت‌های FP32 را استفاده می‌کند.

مزیت آن این است ک کاهش مصرف حافظه داریم. تقریباً نصف حالت FP32 مصرف می‌کند. و افزایش سرعت محاسبات به دلیل استفاده کمتر از منابع سخت‌افزاری. البته ممکن است باعث کاهش دقت محاسبات یا ناپایداری در گرادیان‌ها شود.

bf16یا BFloat 16 :

مشابه FP16، اما با دقت بالاتر در بخش نمایی، که باعث افزایش پایداری محاسبات می‌شود. باتوجه به توضیحات ارائه شده در داکیومنتی که لینکش در تمرین قرار داشت، این حالت تنها بر روی سخت‌افزارهایی مانند Nvidia Ampere GPUs و PyTorch 1.10+ پشتیبانی می‌شود. مزیت آن کاهش مشکلات ناپایداری در مقایسه با FP16 و مصرف حافظه و سرعت محاسبات مشابه FP16 است.

fp8یا Float 8 :

یک حالت جدید که محاسبات را با دقت بسیار پایین‌تر انجام می‌دهد.البته این حالت هنوز به طور گسترده پشتیبانی نمی‌شود و برای سخت‌افزارهای خاصی طراحی شده است و خب محدودیت اصلی این حالت در دسترس نبودن سخت‌افزار مناسب برای اجرا است و ماشین‌های کلاستر درس هم پشتیبانی نمی‌کنند.

۲-۳_ آموزش مدل با mixed precision

نتایج حالت Mixed Precision با مقدار FP16 (یک ماشین و یک هسته)

```
----------------------
Nodelist:  raspberrypi-dml0
Number of nodes:  1
Ntasks per node:  1
CPUs per task: 1
 ----------------------
MASTER_ADDR: raspberrypi-dml0:24442
----------------------
DistributedType.NO
Training Started for Rank = 0
Epoch [1/10], Train Loss: 1.0441, Train Accuracy: 72.87%, Test Loss: 0.5069, Test Accuracy:
84.74%
Epoch [2/10], Train Loss: 0.5082, Train Accuracy: 84.11%, Test Loss: 0.4651, Test Accuracy:
85.23%
Epoch [3/10], Train Loss: 0.4657, Train Accuracy: 85.14%, Test Loss: 0.4537, Test Accuracy:
85.60%
Epoch [4/10], Train Loss: 0.4372, Train Accuracy: 85.99%, Test Loss: 0.4507, Test Accuracy:
85.69%
Epoch [5/10], Train Loss: 0.4248, Train Accuracy: 86.30%, Test Loss: 0.4509, Test Accuracy:
85.94%
Epoch [6/10], Train Loss: 0.4127, Train Accuracy: 86.83%, Test Loss: 0.4536, Test Accuracy:
85.39%
Epoch [7/10], Train Loss: 0.4012, Train Accuracy: 86.99%, Test Loss: 0.4516, Test Accuracy:
85.82%
Epoch [8/10], Train Loss: 0.3947, Train Accuracy: 87.28%, Test Loss: 0.4570, Test Accuracy:
85.65%
```

```
Epoch [9/10], Train Loss: 0.3865, Train Accuracy: 87.42%, Test Loss: 0.4493, Test Accuracy:
86.06%
Epoch [10/10], Train Loss: 0.3832, Train Accuracy: 87.56%, Test Loss: 0.4553, Test Accuracy:
85.67%
Final Train Accuracy: 87.56%
Final Test Accuracy: 85.67%
Total Training Time: 183.36 seconds
```

دقت آموزش در این حالت به **٪۸۷,۵۶** رسید، که نشان‌دهنده یادگیری خوب مدل با استفاده از دقت FP16 است.دقت تست برابر با **٪۸۵,۶۷** بود، که مشابه با حالت‌های دیگر و بدون افت دقت قابل توجه است. زمان کل آموزش **۱۸۳,۳۶ ثانیه** بود.

نتایج حالت Mixed Precision با مقدار BF16 (یک ماشین و یک هسته)

دقت آموزش در این حالت به **٪۸۷,۵۷** رسید و دقت تست به **٪۸۵,۶۶** بود. زمان کل آموزش در این حالت **۲۱۰,۷۲** ثانیه بود که نسبت به FP16 و FP32 طولانی‌تر بود. این نتیجه نشان می‌دهد که در سیستم‌های بدون سخت‌افزار پیشرفته، استفاده از BF16 ممکن است کارآمد نباشد.

Mixed Precision با مقدار FP16 (یک ماشین و دو هسته)

```
----------------------
Nodelist:  raspberrypi-dml0
Number of nodes:  1
Ntasks per node:  1
CPUs per task: 2
 ----------------------
MASTER_ADDR: raspberrypi-dml0:24442
----------------------
W0117 04:15:28.513000 1607564 torch/distributed/run.py:785] master_addr is only used for static
rdzv_backend and when rdzv_endpoint is not specified.
DistributedType.MULTI_CPU
Training Started for Rank = 0
Training Started for Rank = 1
Epoch [1/10], Train Loss: 1.2321, Train Accuracy: 70.07%, Test Loss: 0.5478, Test Accuracy:
84.22%
Epoch [2/10], Train Loss: 0.5222, Train Accuracy: 84.35%, Test Loss: 0.4639, Test Accuracy:
84.97%
Epoch [3/10], Train Loss: 0.4628, Train Accuracy: 85.54%, Test Loss: 0.4601, Test Accuracy:
85.47%
Epoch [4/10], Train Loss: 0.4344, Train Accuracy: 86.17%, Test Loss: 0.4577, Test Accuracy:
85.17%
Epoch [5/10], Train Loss: 0.4168, Train Accuracy: 86.69%, Test Loss: 0.4561, Test Accuracy:
85.63%
Epoch [6/10], Train Loss: 0.4068, Train Accuracy: 87.00%, Test Loss: 0.4523, Test Accuracy:
85.43%
Epoch [7/10], Train Loss: 0.4036, Train Accuracy: 86.95%, Test Loss: 0.4498, Test Accuracy:
85.35%
Epoch [8/10], Train Loss: 0.3951, Train Accuracy: 87.38%, Test Loss: 0.4578, Test Accuracy:
85.43%
Epoch [9/10], Train Loss: 0.3903, Train Accuracy: 87.18%, Test Loss: 0.4546, Test Accuracy:
85.47%
Epoch [10/10], Train Loss: 0.3747, Train Accuracy: 87.68%, Test Loss: 0.4540, Test Accuracy:
85.51%
Final Train Accuracy: 87.68%
Final Test Accuracy: 85.51%
Total Training Time: 177.68 seconds
```

دقت آموزش به **٪۸۷,۶۸** رسید و دقت تست برابر با **٪۸۵,۵۱** بود. برای این شبکه تغییری ایجاد نشد. شاید بخاطر ساده بودن شبکه باشد. زمان کل آموزش **۱۷۷,۶۸ ثانیه** بود. این زمان در مقایسه با

حالت‌های قبلی کمی افزایش داشته است. این موضوع می‌تواند ناشی از سربار مدیریت Mixed Precision یا نبود سخت‌افزار کاملاً بهینه برای FP16 باشد.

نتایج حالت Mixed Precision با مقدار BF16 (یک ماشین و دو هسته)

```
----------------------
Nodelist:  raspberrypi-dml0
Number of nodes:  1
Ntasks per node:  1
CPUs per task: 2
 ----------------------
MASTER_ADDR: raspberrypi-dml0:24442
----------------------
W0117 04:23:33.147000 1607706 torch/distributed/run.py:785] master_addr is only used for static
rdzv_backend and when rdzv_endpoint is not specified.
DistributedType.MULTI_CPU
Training Started for Rank = 0Training Started for Rank = 1

Epoch [1/10], Train Loss: 1.2318, Train Accuracy: 70.11%, Test Loss: 0.5482, Test Accuracy:
84.34%
Epoch [2/10], Train Loss: 0.5223, Train Accuracy: 84.26%, Test Loss: 0.4642, Test Accuracy:
84.81%
Epoch [3/10], Train Loss: 0.4625, Train Accuracy: 85.51%, Test Loss: 0.4603, Test Accuracy:
85.23%
Epoch [4/10], Train Loss: 0.4347, Train Accuracy: 86.20%, Test Loss: 0.4569, Test Accuracy:
85.15%
Epoch [5/10], Train Loss: 0.4168, Train Accuracy: 86.71%, Test Loss: 0.4561, Test Accuracy:
85.45%
Epoch [6/10], Train Loss: 0.4070, Train Accuracy: 87.06%, Test Loss: 0.4520, Test Accuracy:
85.27%
Epoch [7/10], Train Loss: 0.4038, Train Accuracy: 86.86%, Test Loss: 0.4507, Test Accuracy:
85.43%
Epoch [8/10], Train Loss: 0.3955, Train Accuracy: 87.36%, Test Loss: 0.4579, Test Accuracy:
85.69%
Epoch [9/10], Train Loss: 0.3906, Train Accuracy: 87.17%, Test Loss: 0.4549, Test Accuracy:
85.25%
Epoch [10/10], Train Loss: 0.3745, Train Accuracy: 87.79%, Test Loss: 0.4542, Test Accuracy:
85.55%
Final Train Accuracy: 87.79%
Final Test Accuracy: 85.55%
Total Training Time: 191.12 seconds
```

دقت آموزش برابر با **۸۷,۷۹٪** بود، که نشان‌دهنده یادگیری مناسب مدل با استفاده از دقت BF16 است. دقت تست به **۸۵,۵۵٪** رسید، که در مقایسه با حالت FP16 کمی بهبود داشت. زمان کل آموزش **۱۹۱,۱۲ ثانیه** بود، که بیشتر از حالت FP16است.

نتایج حالت Mixed Precision با مقدار FP8 (یک ماشین و یک هسته)

اجرای کد با تنظیم FP8 به خطای زیر منجر شد:

```
----------------------
Nodelist:  raspberrypi-dml0
Number of nodes:  1
Ntasks per node:  1
CPUs per task: 1
 ----------------------
MASTER_ADDR: raspberrypi-dml0:24442
----------------------
Traceback (most recent call last):
  File "/home/shared_files/pytorch_venv/bin/accelerate", line 8, in <module<
    sys.exit(main())
^^^^^^
  File                                  "/home/shared_files/pytorch_venv/lib/python3.11/site-
packages/accelerate/commands/accelerate_cli.py", line 48, in main
```

```
    args.func(args)
  File                                    "/home/shared_files/pytorch_venv/lib/python3.11/site-
packages/accelerate/commands/launch.py", line 1168, in launch_command
    simple_launcher(args)
  File                                    "/home/shared_files/pytorch_venv/lib/python3.11/site-
packages/accelerate/commands/launch.py", line 757, in simple_launcher
    cmd, current_env = prepare_simple_launcher_cmd_env(args)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File                                    "/home/shared_files/pytorch_venv/lib/python3.11/site-
packages/accelerate/utils/launch.py", line 158, in prepare_simple_launcher_cmd_env
    raise RuntimeError)
RuntimeError: FP8 is not available on this machine. Please ensure that either Transformer Engine
or MSAMP is installed.
srun: error: raspberrypi-dml0: task 0: Exited with exit code 1
```

این خطا نشان می‌دهد که سخت‌افزار موجود از دقت FP8 پشتیبانی نمی‌کند. کتابخانه‌هایی مانند Transformer Engine و MSAMP ابزارهای مورد نیاز برای اجرای محاسبات FP8 هستند. نصب این کتابخانه‌ها و البته تنها در صورتی مفید است که سخت‌افزار مناسبی در دسترس باشد.

۳-۳_ تحلیل نتایج

باتوجه به آزمایش‌هاییکه انجام دادیم متوجه شدیم که Mixed Precision در حالت‌های FP16 و BF16 می‌تواند دقت مدل را حفظ کند و در شرایط مناسب باعث کاهش زمان و مصرف حافظه شود. با این حال، استفاده از آن به سخت‌افزار و نرم‌افزار مناسب بستگی دارد. برای سیستم‌های عمومی، FP16 گزینه‌ای کارآمدتر است، در حالی که FP8 تنها برای سخت‌افزارهای پیشرفته قابل استفاده است.

حالت	دقت آموزش	دقت تست	زمان آموزش (ثانیه)
FP16(۱ ماشین، ۱ هسته)	۸۷,۵۶	۸۵,۶۷	۱۸۳,۳۶
BF16(۱ ماشین، ۱ هسته)	۸۷,۵۷	۸۵,۶۶	۲۱۰,۷۲
FP16(۱ ماشین، ۲ هسته)	۸۷,۶۸	۸۵,۵۱	۱۷۷,۶۸
BF16(۱ ماشین، ۲ هسته)	۸۷,۷۹	۸۵,۵۵	۱۹۱,۱۲
FP8	پشتیبانی نشد	پشتیبانی نشد	پشتیبانی نشد

جدول ۳ نتایج آزمایش با کمک Accelerate و تنظیم Mixed Precision

روش Accelerate ساده در حالت چندماشینی عملکرد بهتری در کاهش زمان آموزش داشت. برای مثال، در حالت دو ماشین و دو هسته، زمان آموزش با Accelerate به ۱۱۳,۵۵ ثانیه رسید، در حالی که Mixed Precision در حالت FP16 یا BF16 زمان بیشتری صرف کرد.

۴_ پاسخ سوال شماره ۴

برای پاسخ به این سوال از محیط Kaggle استفاده کردیم. پس از بارگذاری کتابخانه‌ها از همان کد مدل قبلی استفاده کردیم. مدل شامل دو لایه خطی (Linear)، یک لایه نرمال‌سازی (BatchNorm1d)، و یک تابع فعال‌سازی (ReLU) است. داده‌های تست با استفاده از DataLoader در دسته‌های ۳۲تایی بارگذاری می‌شوند. من منظور سوال را دقیق متوجه نشدم که آیا دیتالودر باید بچ ۱۰۰ تایی باشد و یا اینکه ۱۰۰ بچ دیتا را به مدل بدهیم و سپس نتایج را بسنجیم. وقتی که دیتالودر را ۱۰۰ تایی گذاشتم (مرسوم نیست توان ۲ نباشد) زمان CPU بیشتر شد. بهرحال هر دو حالت خروجی گرفته شد.

```
input_size = 512
hidden_size = 32
output_size = 20
model = SimpleModel(input_size, hidden_size, output_size).to("cpu")
test_loader = load_test_data()
```

مدل را بارگذاری کردیم.

باتوجه به توضیحات داکیومنت معرفی شده در فایل تمرین هم حالت فقط CPU و هم حالت CPUو MEMORY اجرا شد. باتوجه به اینکه حالت دوم تمامی موارد اولی را در برمیگیرد از توضیح اولی صرف نظر می‌کنیم.

۴-۱_ راه‌اندازی Profiler

کد زیر برای راه‌اندازی استفاده شد:

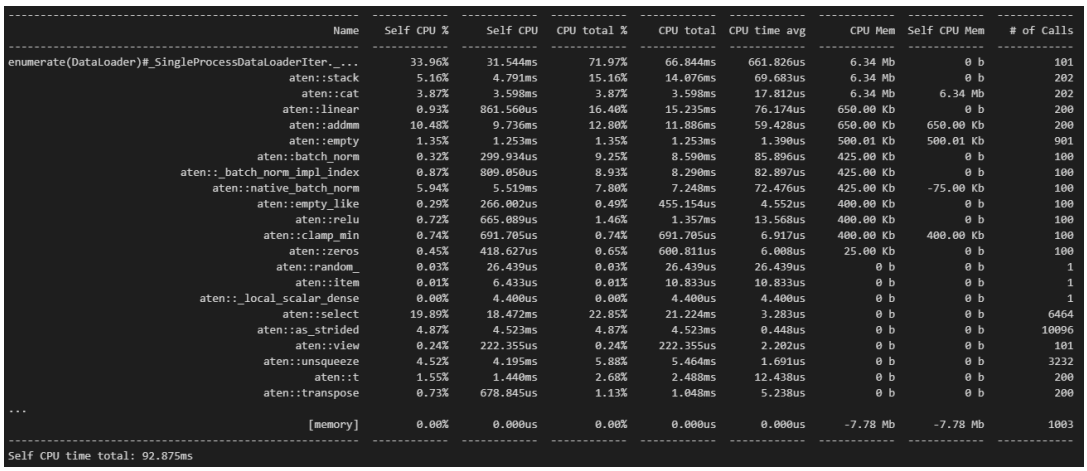
```
with  profile(activities=[ProfilerActivity.CPU],profile_memory=True,  record_shapes=True)  as
prof:
    for i, (inputs, _) in enumerate(test_loader):
      if i >= 100:  # 100 batches
        break
      model(inputs)
```

activities مشخص می‌کند که پروفایلر کدام مورد را ثبت کند. در اینجا تنها فعالیت‌های CPU بررسی شده و profile_memory=True حافظه مصرفی را برای هر عملیات ثبت می‌کند.

```
for i, (inputs, _) in enumerate(test_loader):
  if i >= 100:  # 100 batches
    break
  model(inputs)
```

حلقه برای پردازش ۱۰۰ دسته از داده‌های تست استفاده شد. key_averages: عملیات‌های پروفایلر را تجمیع کرده و نتایج را به صورت خلاصه جدولی نمایش می‌دهد.

نتیجه خروجی جدول زیر بود:



Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	CPU Mem	Self CPU Mem	# of Calls
enumerate(DataLoader)#_SingleProcessDataLoaderIter._...	33.96%	31.544ms	71.97%	66.844ms	661.826us	6.34 Mb	0 b	101
aten::stack	5.16%	4.791ms	15.16%	14.076ms	69.683us	6.34 Mb	0 b	202
aten::cat	3.87%	3.598ms	3.87%	3.598ms	17.812us	6.34 Mb	6.34 Mb	202
aten::linear	0.93%	861.560us	16.40%	15.235ms	76.174us	650.00 Kb	0 b	200
aten::addmm	10.48%	9.736ms	12.80%	11.886ms	59.428us	650.00 Kb	650.00 Kb	200
aten::empty	1.35%	1.253ms	1.35%	1.253ms	1.390us	500.01 Kb	500.01 Kb	901
aten::batch_norm	0.32%	299.934us	9.25%	8.590ms	85.896us	425.00 Kb	0 b	100
aten::_batch_norm_impl_index	0.87%	809.050us	8.93%	8.290ms	82.897us	425.00 Kb	0 b	100
aten::native_batch_norm	5.94%	5.519ms	7.80%	7.248ms	72.476us	425.00 Kb	-75.00 Kb	100
aten::empty_like	0.29%	266.002us	0.49%	455.154us	4.552us	400.00 Kb	0 b	100
aten::relu	0.72%	665.089us	1.46%	1.357ms	13.568us	400.00 Kb	0 b	100
aten::clamp_min	0.74%	691.705us	0.74%	691.705us	6.917us	400.00 Kb	400.00 Kb	100
aten::zeros	0.45%	418.627us	0.65%	600.811us	6.008us	25.00 Kb	0 b	100
aten::random_	0.03%	26.439us	0.03%	26.439us	26.439us	0 b	0 b	1
aten::item	0.01%	6.433us	0.01%	10.833us	10.833us	0 b	0 b	1
aten::_local_scalar_dense	0.00%	4.400us	0.00%	4.400us	4.400us	0 b	0 b	1
aten::select	19.89%	18.472ms	22.85%	21.224ms	3.283us	0 b	0 b	6464
aten::as_strided	4.87%	4.523ms	4.87%	4.523ms	0.448us	0 b	0 b	10096
aten::view	0.24%	222.355us	0.24%	222.355us	2.202us	0 b	0 b	101
aten::unsqueeze	4.52%	4.195ms	5.88%	5.464ms	1.691us	0 b	0 b	3232
aten::t	1.55%	1.440ms	2.68%	2.488ms	12.438us	0 b	0 b	200
aten::transpose	0.73%	678.845us	1.13%	1.048ms	5.238us	0 b	0 b	200
...								
[memory]	0.00%	0.000us	0.00%	0.000us	0.000us	-7.78 Mb	-7.78 Mb	1003

Self CPU time total: 92.875ms

شکل ۴-۱ خروجی Profiler پایتورچ به صورت جدول

۴-۲_ مقایسه زمان و حافظه مصرفی ماژول‌ها

زمان اجرا:

Linear: این ماژول بیشترین زمان اجرا را مصرف کرده است. دلیل این احتمالا تعداد عملیات محاسباتی بالای آن در تنسورهای بزرگ است.

BatchNorm: دومین ماژول از نظر مصرف زمان است. این ماژول به دلیل نیاز به محاسبات میانگین و واریانس بر روی هر دسته زمان بیشتری نسبت به توابع ساده مانند ReLU مصرف می‌کند.

ReLU: کمترین زمان اجرا را دارد، زیرا تنها عملیات ساده‌ای مانند مقایسه و تغییر مقادیر منفی به صفر انجام می‌دهد.

ماژول	زمان کل	حافظه مصرفی کل	تعداد فراخوانی
Linear	15.235ms	650.00 Kb	200
BatchNorm	8.590ms	425.00 Kb	100
ReLU	1.357ms	400.00 Kb	100

جدول ۴ جدول مقایسه زمان و حافظه مصرفی ماژول‌های Linear، BatchNorm، و ReLU

مصرف حافظه:

Linear: با مصرف ۶۵۰ کیلوبایت حافظه بیشترین سهم را دارد، زیرا نیاز به ذخیره وزن‌ها و نتایج محاسبات دارد.

BatchNorm: حافظه مصرفی آن ۴۲۵ کیلوبایت است که کمتر از Linear است. این حافظه برای ذخیره میانگین‌ها و واریانس‌ها استفاده می‌شود.

ReLU: حافظه مصرفی آن ۴۰۰ کیلوبایت است که کمترین مقدار بین همه است. این حافظه تنها برای ذخیره مقادیر خروجی فعال‌سازی است.

مقایسه زمان و حافظه توابع فعال‌سازی: ReLU, Tanh, Sigmoid, GeLU

برای این بخش هربار مدل را تغییر داده و در بلوک مربوطه در نوت بوک خروجی گرفتیم. نتایج به صورت زیر است:

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	CPU Mem	Self CPU Mem	# of Calls
enumerate(DataLoader)#_SingleProcessDataLoaderIter._...	33.00%	29.237ms	72.03%	63.818ms	631.863us	6.34 Mb	0 b	101
aten::stack	4.97%	4.402ms	15.19%	13.455ms	66.610us	6.34 Mb	0 b	202
aten::cat	3.95%	3.500ms	3.95%	3.500ms	17.328us	6.34 Mb	6.34 Mb	202
aten::linear	1.07%	949.353us	16.24%	14.391ms	71.953us	650.00 Kb	0 b	200
aten::addmm	10.17%	9.013ms	12.54%	11.109ms	55.543us	650.00 Kb	650.00 Kb	200
aten::empty	1.36%	1.208ms	1.36%	1.208ms	1.340us	500.01 Kb	500.01 Kb	901
aten::batch_norm	0.31%	274.470us	8.95%	7.927ms	79.271us	425.00 Kb	0 b	100
aten::_batch_norm_impl_index	0.83%	735.816us	8.64%	7.653ms	76.526us	425.00 Kb	0 b	100
aten::native_batch_norm	5.69%	5.043ms	7.60%	6.733ms	67.327us	425.00 Kb	-75.00 Kb	100
aten::empty_like	0.29%	256.510us	0.47%	417.668us	4.177us	400.00 Kb	0 b	100
aten::tanh	1.97%	1.749ms	1.97%	1.749ms	17.486us	400.00 Kb	400.00 Kb	100
aten::zeros	0.41%	365.523us	0.63%	555.389us	5.554us	25.00 Kb	0 b	100
aten::random_	0.02%	13.571us	0.02%	13.571us	13.571us	0 b	0 b	1
aten::item	0.01%	7.068us	0.01%	12.434us	12.434us	0 b	0 b	1
aten::_local_scalar_dense	0.01%	5.366us	0.01%	5.366us	5.366us	0 b	0 b	1
aten::select	20.84%	18.466ms	23.84%	21.126ms	3.268us	0 b	0 b	6464
aten::as_strided	4.86%	4.307ms	4.86%	4.307ms	0.427us	0 b	0 b	10096
aten::view	0.23%	199.364us	0.23%	199.364us	1.974us	0 b	0 b	101
aten::unsqueeze	4.70%	4.161ms	6.04%	5.353ms	1.656us	0 b	0 b	3232
aten::t	1.54%	1.362ms	2.63%	2.333ms	11.663us	0 b	0 b	200
aten::transpose	0.72%	641.829us	1.10%	971.030us	4.855us	0 b	0 b	200
aten::expand	0.68%	604.147us	0.82%	729.734us	3.649us	0 b	0 b	200
...								
[memory]	0.00%	0.000us	0.00%	0.000us	0.000us	-7.78 Mb	-7.78 Mb	1003

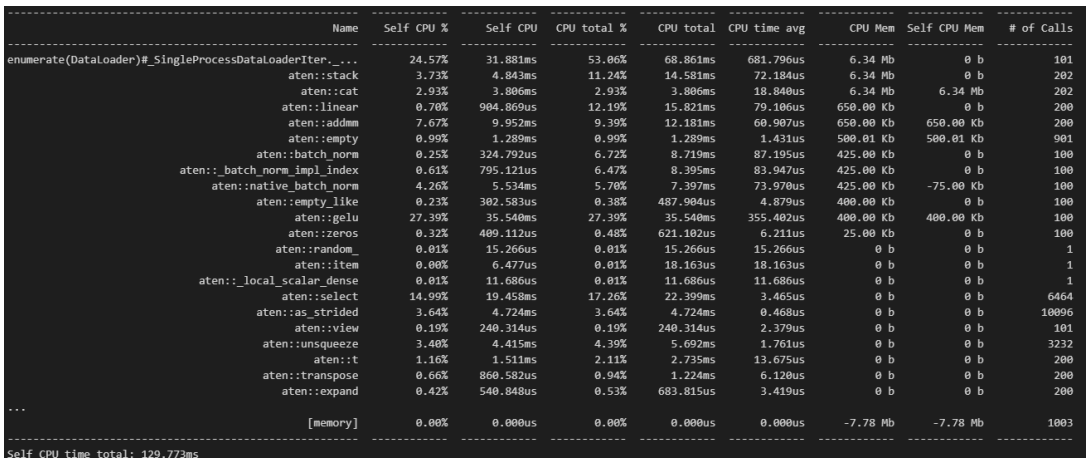
Self CPU time total: 88.602ms

شکل ۴-۲ خروجی Profiler پایتورچ برای مدل با فعال‌ساز Tanh

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	CPU Mem	Self CPU Mem	# of Calls
enumerate(DataLoader)#_SingleProcessDataLoaderIter._...	32.44%	32.156ms	69.44%	68.821ms	681.401us	6.34 Mb	0 b	101
aten::stack	4.99%	4.947ms	15.35%	15.211ms	75.302us	6.34 Mb	0 b	202
aten::cat	3.88%	3.846ms	3.88%	3.846ms	19.040us	6.34 Mb	6.34 Mb	202
aten::linear	0.97%	959.738us	16.36%	16.210ms	81.049us	650.00 Kb	0 b	200
aten::addmm	10.09%	9.996ms	12.60%	12.486ms	62.429us	650.00 Kb	650.00 Kb	200
aten::empty	1.29%	1.278ms	1.29%	1.278ms	1.418us	500.01 Kb	500.01 Kb	901
aten::batch_norm	0.32%	317.170us	9.24%	9.159ms	91.590us	425.00 Kb	0 b	100
aten::_batch_norm_impl_index	0.96%	946.753us	8.92%	8.842ms	88.419us	425.00 Kb	0 b	100
aten::native_batch_norm	5.85%	5.801ms	7.74%	7.670ms	76.697us	425.00 Kb	-75.00 Kb	100
aten::empty_like	0.31%	309.331us	0.49%	489.375us	4.894us	400.00 Kb	0 b	100
aten::sigmoid	4.08%	4.039ms	4.08%	4.039ms	40.395us	400.00 Kb	400.00 Kb	100
aten::zeros	0.45%	450.082us	0.67%	660.232us	6.602us	25.00 Kb	0 b	100
aten::random_	0.01%	13.100us	0.01%	13.100us	13.100us	0 b	0 b	1
aten::item	0.00%	4.516us	0.01%	7.713us	7.713us	0 b	0 b	1
aten::_local_scalar_dense	0.00%	3.197us	0.00%	3.197us	3.197us	0 b	0 b	1
aten::select	18.87%	18.700ms	21.65%	21.455ms	3.319us	0 b	0 b	6464
aten::as_strided	4.79%	4.752ms	4.79%	4.752ms	0.471us	0 b	0 b	10096
aten::view	0.24%	234.299us	0.24%	234.299us	2.320us	0 b	0 b	101
aten::unsqueeze	4.72%	4.677ms	6.24%	6.184ms	1.913us	0 b	0 b	3232
aten::t	1.70%	1.683ms	2.79%	2.764ms	13.821us	0 b	0 b	200
aten::transpose	0.74%	729.832us	1.09%	1.081ms	5.406us	0 b	0 b	200
aten::expand	0.68%	671.363us	0.82%	809.959us	4.050us	0 b	0 b	200
...								
[memory]	0.00%	0.000us	0.00%	0.000us	0.000us	-7.78 Mb	-7.78 Mb	1003

Self CPU time total: 99.110ms

شکل ۴-۳ خروجی Profiler پایتورچ برای مدل با فعال‌ساز Sigmoid



Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	CPU Mem	Self CPU Mem	# of Calls
enumerate(DataLoader)#_SingleProcessDataLoaderIter._...	24.57%	31.881ms	53.06%	68.861ms	681.796us	6.34 Mb	0 b	101
aten::stack	3.73%	4.843ms	11.24%	14.581ms	72.184us	6.34 Mb	0 b	202
aten::cat	2.93%	3.806ms	2.93%	3.806ms	18.840us	6.34 Mb	6.34 Mb	202
aten::linear	0.70%	904.869us	12.19%	15.821ms	79.106us	650.00 Kb	0 b	200
aten::addmm	7.67%	9.952ms	9.39%	12.181ms	60.907us	650.00 Kb	650.00 Kb	200
aten::empty	0.99%	1.289ms	0.99%	1.289ms	1.431us	500.01 Kb	500.01 Kb	901
aten::batch_norm	0.25%	324.792us	6.72%	8.719ms	87.195us	425.00 Kb	0 b	100
aten::_batch_norm_impl_index	0.61%	795.121us	6.47%	8.395ms	83.947us	425.00 Kb	0 b	100
aten::native_batch_norm	4.26%	5.534ms	5.70%	7.397ms	73.970us	425.00 Kb	-75.00 Kb	100
aten::empty_like	0.23%	302.583us	0.38%	487.904us	4.879us	400.00 Kb	0 b	100
aten::gelu	27.39%	35.540ms	27.39%	35.540ms	355.402us	400.00 Kb	400.00 Kb	100
aten::zeros	0.32%	409.112us	0.48%	621.102us	6.211us	25.00 Kb	0 b	100
aten::random_	0.01%	15.266us	0.01%	15.266us	15.266us	0 b	0 b	1
aten::item	0.00%	6.477us	0.01%	18.163us	18.163us	0 b	0 b	1
aten::_local_scalar_dense	0.01%	11.686us	0.01%	11.686us	11.686us	0 b	0 b	1
aten::select	14.99%	19.458ms	17.26%	22.399ms	3.465us	0 b	0 b	6464
aten::as_strided	3.64%	4.724ms	3.64%	4.724ms	0.468us	0 b	0 b	10096
aten::view	0.19%	240.314us	0.19%	240.314us	2.379us	0 b	0 b	101
aten::unsqueeze	3.40%	4.415ms	4.39%	5.692ms	1.761us	0 b	0 b	3232
aten::t	1.16%	1.511ms	2.11%	2.735ms	13.675us	0 b	0 b	200
aten::transpose	0.66%	860.582us	0.94%	1.224ms	6.120us	0 b	0 b	200
aten::expand	0.42%	540.848us	0.53%	683.815us	3.419us	0 b	0 b	200
...								
[memory]	0.00%	0.000us	0.00%	0.000us	0.000us	-7.78 Mb	-7.78 Mb	1003

Self CPU time total: 129.773ms

شکل ۴-۴ خروجی Profiler پایتورچ برای مدل با فعال‌ساز GeLU

تحلیل نتایج

جدول مقایسه به صورت زیر است :

تابع فعال‌سازی	زمان کل	حافظه مصرفی کل	تعداد فراخوانی
ReLU	1.357ms	400.00 Kb	100
Tanh	1.749ms	400.00 Kb	100
Sigmoid	4.039ms	400.00 Kb	100
GeLU	35.540ms	400.00 Kb	100

جدول ۵ مقایسه زمان و حافظه مصرفی توابع فعال‌سازی: ReLU, Tanh, Sigmoid, GeLU

ReLU ساده‌ترین و سریع‌ترین تابع فعال‌سازی در میان توابع مورد بررسی است. زمان اجرای آن تنها ۱,۳۵۷ میلی‌ثانیه بود.عملیات اصلی آن شامل مقایسه و صفر کردن مقادیر منفی است که باعث کاهش زمان محاسبات می‌شود.

Tanh با زمان اجرای ۱,۷۴۹ میلی‌ثانیه، کمی کندتر از ReLU است.

Sigmoid زمان اجرای آن ۴,۰۳۹ میلی‌ثانیه بود، که نشان‌دهنده پیچیدگی بیشتر آن نسبت به ReLU و Tanh است. تابع Sigmoid برای محاسبه خروجی نیاز به محاسبات نمایی دارد.

GeLU با زمان اجرای ۳۵,۵۴۰ میلی‌ثانیه، کندترین تابع فعال‌سازی است. دلیل این زمان طولانی، محاسبات پیچیده‌تر مربوط به تقریب توزیع گاوسی است.

تمامی توابع فعال‌سازی مقدار مشابهی حافظه بیرابر ۴۰۰ کیلوبایت مصرف کردند. این حافظه برای ذخیره مقادیر خروجی فعال‌سازی استفاده می‌شود و تفاوتی در مصرف حافظه میان این توابع مشاهده نشد.

۵_ ابزارهای استفاده شده

برای نگارش این تمرین از ابزارهای مبتنی بر مدل زبانی و همچنین ابزار پاک‌نویس برای رعایت نیم‌فاصله و علائم نگارشی استفاده شد.

پرامپت‌های مهم استفاده شده :

یک مدل ساده برای PyTorch تعریف کنید که شامل BatchNorm، ReLU، و لایه‌های Linear باشد.

Mixed Precision چیست و چگونه می‌توان آن را با کتابخانه Accelerate فعال کرد؟

برای نتایج زیر جدول بسازید. (تمام جداول را به این صورت ساختم که نتایج را دادم و درخواست جدول می‌کردم)

چگونه می‌توان با استفاده از Accelerate و Slurm، مدل را به صورت توزیع‌شده روی چند ماشین آموزش داد؟

من کد زیر را زده ام ولی هر ماشین آموزش را جداگانه انجام می‌دهد. (به دنبال راه حل اجرای توزیع شده در Accelerate)

چگونه می‌توان تعداد هسته‌ها و ماشین‌ها را در Slurm برای Accelerate تنظیم کرد؟

دستور کامل srun accelerate launch را برای اجرای مدل روی دو ماشین و چهار هسته بنویسید.

چگونه می‌توان با استفاده از Accelerate یک اسکریپت پایتون ساده نوشت که مدل را به صورت توزیع‌شده اجرا کند؟

چگونه شماره هر فرایند توزیع شده را در Accelerate خروجی بگیریم.