pytorch - GPU

Lecture 19

Dr. Colin Rundel

CUDA

> CUDA (or Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) that allows software to use certain types of graphics processing unit (GPU) for general purpose processing, an approach called general-purpose computing on GPUs (GPGPU). CUDA is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.

Core libraries:

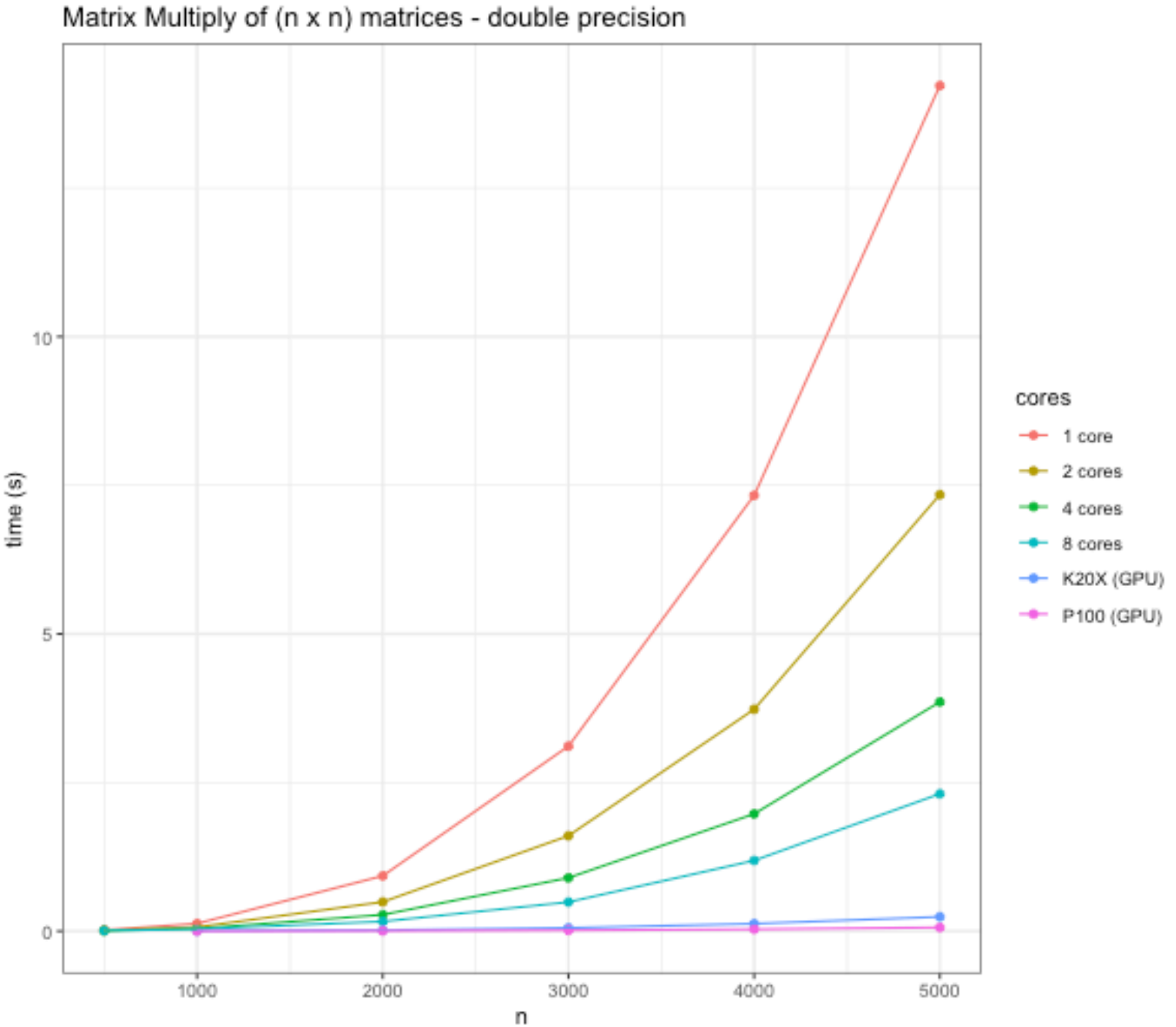
- cuBLAS
- cuSOLVER
- cuSPARSE
- cuTENSOR
- cuFFT
- cuRAND
- Thrust
- cuDNN

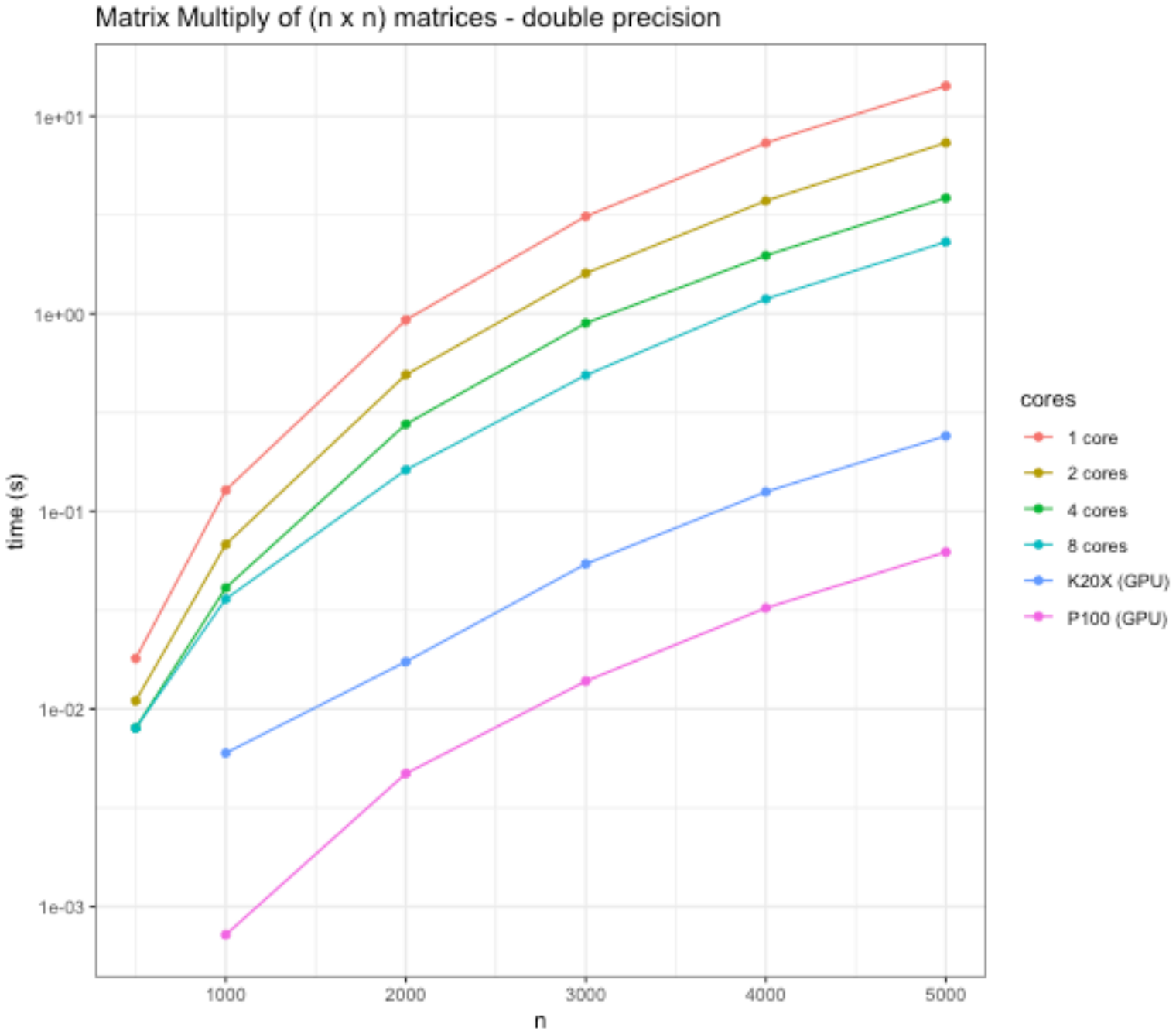
CUDA Kernels

```
// Kernel - Adding two matrices MatA and MatB
__global__ void MatAdd(float MatA[N][N], float MatB[N][N], float MatC[N][N])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        MatC[i][j] = MatA[i][j] + MatB[i][j];
}

int main()
{
    ...
    // Matrix addition kernel launch from host code
    dim3 threadsPerBlock(16, 16);
    dim3 numBlocks(
        (N + threadsPerBlock.x -1) / threadsPerBlock.x,
        (N+threadsPerBlock.y -1) / threadsPerBlock.y
    );

    MatAdd<<<numBlocks, threadsPerBlock>>>(MatA, MatB, MatC);
    ...
}
```


Matrix Multiply of (n x n) matrices - double precision
time (s)
0
5
10
1000
2000
3000
4000
5000
n
cores
1 core
2 cores
4 cores
8 cores
K20X (GPU)
P100 (GPU)


Matrix Multiply of (n x n) matrices - double precision
time (s)
1e+01
1e+00
1e-01
1e-02
1e-03
1000
2000
3000
4000
5000
n
cores
1 core
2 cores
4 cores
8 cores
K20X (GPU)
P100 (GPU)

GPU Status

```
1 nvidia-smi
```

```
Tue Mar 25 13:53:59 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 565.57.01              Driver Version: 565.57.01      CUDA Version: 12.7     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA RTX A4000               Off |   00000000:01:00.0 Off |                  Off |
| 41%   36C    P8             15W /  140W |     915MiB /  16376MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   1  NVIDIA RTX A4000               Off |   00000000:68:00.0 Off |                  Off |
| 41%   37C    P8             11W /  140W |       4MiB /  16376MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
```

Torch GPU Information

```
1 torch.cuda.is_available()
```

```
True
```

```
1 torch.cuda.device_count()
```

```
2
```

```
1 torch.cuda.get_device_name("cuda:0")
```

```
'NVIDIA RTX A4000'
```

```
1 torch.cuda.get_device_name("cuda:1")
```

```
'NVIDIA RTX A4000'
```

```
1 torch.cuda.get_device_properties(0)
```

```
_CudaDeviceProperties(name='NVIDIA RTX A4000', major=8, minor=6, total_memory=16001MB, multi_processor_count=48,
   uuid=4437a926-fb7d-d9b4-ff50-f16792d2b088, L2_cache_size=4MB)
```

```
1 torch.cuda.get_device_properties(1)
```

```
_CudaDeviceProperties(name='NVIDIA RTX A4000', major=8, minor=6, total_memory=16000MB, multi_processor_count=48,
   uuid=13910be2-a0c4-69fe-7704-31f6448209d9, L2_cache_size=4MB)
```

GPU Tensors

Usage of the GPU is governed by the location of the Tensors - to use the GPU we allocate them on the GPU device.

```
1  cpu = torch.device('cpu')
2  cuda0 = torch.device('cuda:0')
3  cuda1 = torch.device('cuda:1')
4
5  x = torch.linspace(0,1,5, device=cuda0); x
```

```
tensor([0.0000, 0.2500, 0.5000, 0.7500,
        1.0000], device='cuda:0')
```

```
1  y = torch.randn(5,2, device=cuda0); y
```

```
tensor([[-0.2178, -0.3361],
        [-0.3228, -0.0225],
        [-0.7392, -0.0719],
        [-0.2632, -0.4953],
        [-0.0131, -0.1414]], device='cuda:0')
```

```
1  z = torch.rand(2,3, device=cpu); z
```

```
tensor([[0.1798, 0.0415, 0.7274],
        [0.5587, 0.8566, 0.6186]])
```

```
1  x @ y
```

```
tensor([-0.6608, -0.5545], device='cuda:0')
```

```
1  y @ z
```

```
RuntimeError: Expected all tensors to be on the same
device, but found at least two devices, cuda:0 and cpu!
(when
   checking argument for argument mat2 in method
wrapper_CUDA_mm)
```

```
1  y @ z.to(cuda0)
```

```
tensor([[-0.2269, -0.2970, -0.3664],
        [-0.0706, -0.0327, -0.2487],
        [-0.1731, -0.0923, -0.5822],
        [-0.3241, -0.4352, -0.4979],
        [-0.0814, -0.1217, -0.0970]],
       device='cuda:0')
```

NN Layers + GPU

NN layers (parameters) also need to be assigned to the GPU to be used with GPU tensors,

```
1  nn = torch.nn.Linear(5,5)
2  X = torch.randn(10,5).cuda()
```

```
1  nn(X)
```

```
RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cpu and cuda:0! (when
   checking argument for argument mat1 in method wrapper_CUDA_addmm)
```

Back to MNIST

Same MNIST data from last time (1x8x8 images),

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

digits = load_digits()
X, y = digits.data, digits.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, shuffle=True, random_state=1234
)

X_train = torch.from_numpy(X_train).float()
y_train = torch.from_numpy(y_train)
X_test = torch.from_numpy(X_test).float()
y_test = torch.from_numpy(y_test)
```

To use the GPU for computation we need to copy these tensors to the GPU,

```
X_train_cuda = X_train.to(device=cuda0)
y_train_cuda = y_train.to(device=cuda0)
X_test_cuda = X_test.to(device=cuda0)
y_test_cuda = y_test.to(device=cuda0)
```

Convolutional NN

```
class mnist_conv_model(torch.nn.Module):
    def __init__(self, device):
        super().__init__()
        self.device = torch.device(device)

        self.model = torch.nn.Sequential(
          torch.nn.Unflatten(1, (1,8,8)),
          torch.nn.Conv2d(
            in_channels=1, out_channels=8,
            kernel_size=3, stride=1, padding=1
          ),
          torch.nn.ReLU(),
          torch.nn.MaxPool2d(kernel_size=2),
          torch.nn.Flatten(),
          torch.nn.Linear(8 * 4 * 4, 10)
        ).to(device=self.device)

    def forward(self, X):
        return self.model(X)

    def fit(self, X, y, lr=0.001, n=1000, acc_step=10):
      opt = torch.optim.SGD(self.parameters(), lr=lr, momentum=0.9)
      losses = []
      for i in range(n):
          opt.zero_grad()
          loss = torch.nn.CrossEntropyLoss()(self(X), y)
```

CPU vs Cuda

```
m = mnist_conv_model(device="cpu")
loss = m.fit(X_train, y_train, n=1000)
loss[-1]
```

```
0.03082713671028614
```

```
m.accuracy(X_test, y_test)
```

```
tensor(0.9806)
```

```
m_cuda = mnist_conv_model(device="cuda")
loss = m_cuda.fit(X_train_cuda, y_train_cuda, n=1000)
loss[-1]
```

```
0.03518570214509964
```

```
m_cuda.accuracy(X_test_cuda, y_test_cuda)
```

```
tensor(0.9611, device='cuda:0')
```

Why are the answers here different?

```
X_train.dtype
```

```
torch.float32
```

```
X_train_cuda.dtype
```

```
torch.float32
```

Performance

CPU performance:

```
m = mnist_conv_model(device="cpu")

start = torch.cuda.Event(enable_timing=True)
end = torch.cuda.Event(enable_timing=True)

start.record()
loss = m.fit(X_train, y_train, n=1000)
end.record()

torch.cuda.synchronize()
print(start.elapsed_time(end) / 1000)
```

```
1.84163525390625
```

GPU performance:

```
m_cuda = mnist_conv_model(device="cuda")

start = torch.cuda.Event(enable_timing=True)
end = torch.cuda.Event(enable_timing=True)

start.record()
loss = m_cuda.fit(X_train_cuda, y_train_cuda, n=1000)
end.record()

torch.cuda.synchronize()
print(start.elapsed_time(end) / 1000)
```

```
0.445146728515625
```

Profiling CPU - 1 forward step

```
m = mnist_conv_model(device="cpu")
with torch.autograd.profiler.profile(with_stack=True, profile_memory=True) as prof_cpu:
    tmp = m(X_train)
```

```
print(prof_cpu.key_averages().table(sort_by='self_cpu_time_total', row_limit=10))
```

```
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------
                           Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg       CPU Mem  Self CPU Mem    # of Calls
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------
     aten::mkldnn_convolution        51.23%     758.387us        52.15%     772.112us     772.112us       2.81 Mb           0 b             1
                   aten::addmm        20.93%     309.887us        21.84%     323.322us     323.322us      56.13 Kb      56.13 Kb             1
aten::max_pool2d_with_indices        14.87%     220.127us        14.87%     220.127us     220.127us       2.10 Mb       2.10 Mb             1
               aten::clamp_min         5.74%      84.931us         5.74%      84.931us      84.931us       2.81 Mb       2.81 Mb             1
             aten::convolution         1.12%      16.572us        53.85%     797.240us     797.240us       2.81 Mb           0 b             1
               aten::unflatten         0.80%      11.842us         1.40%      20.659us      20.659us           0 b           0 b             1
                    aten::view         0.76%      11.311us         0.76%      11.311us       5.656us           0 b           0 b             2
                   aten::copy_         0.69%      10.279us         0.69%      10.279us      10.279us           0 b           0 b             1
            aten::_convolution         0.58%       8.556us        52.73%     780.668us     780.668us       2.81 Mb           0 b             1
                   aten::empty         0.56%       8.245us         0.56%       8.245us       4.122us       2.81 Mb       2.81 Mb             2
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 1.480ms
```

Profiling GPU - 1 forward step

```
m_cuda = mnist_conv_model(device="cuda")
with torch.autograd.profiler.profile(with_stack=True) as prof_cuda:
    tmp = m_cuda(X_train_cuda)
```

```
print(prof_cuda.key_averages().table(sort_by='self_cpu_time_total', row_limit=10))
```

```
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                           Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
      aten::cudnn_convolution        85.64%       1.175ms        86.65%       1.189ms       1.189ms             1
                  aten::addmm         3.24%      44.475us         4.39%      60.274us      60.274us             1
              cudaLaunchKernel         1.95%      26.771us         1.95%      26.771us       5.354us             5
            aten::_convolution         1.23%      16.870us        89.38%       1.226ms       1.226ms             1
                   aten::add_         1.10%      15.099us         1.37%      18.736us      18.736us             1
             aten::convolution         1.10%      15.049us        90.48%       1.241ms       1.241ms             1
aten::max_pool2d_with_indices         0.86%      11.742us         1.06%      14.477us      14.477us             1
                   aten::view         0.74%      10.139us         0.74%      10.139us       3.380us             3
               aten::clamp_min         0.73%       9.958us         0.94%      12.924us      12.924us             1
               aten::unflatten         0.61%       8.366us         1.13%      15.509us      15.509us             1
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 1.372ms
```

Profiling CPU - fit

```
m = mnist_conv_model(device="cpu")
with torch.autograd.profiler.profile(with_stack=True, profile_memory=True) as prof_cpu:
    losses = m.fit(X_train, y_train, n=1000)
```

```
print(prof_cpu.key_averages().table(sort_by='self_cpu_time_total', row_limit=10))
```

```
---------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------
                                   Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg       CPU Mem  Self CPU Mem    # of Calls
---------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------
                           aten::mm        29.52%     514.326ms        29.53%     514.452ms     257.226us     706.54 Mb     706.54 Mb          2000
                       aten::addmm         16.80%     292.639ms        17.31%     301.628ms     301.628us      54.82 Mb      54.82 Mb          1000
            aten::convolution_backward        11.58%     201.734ms        11.77%     204.999ms     204.999us     312.50 Kb      10.97 Kb          1000
              aten::mkldnn_convolution        10.97%     191.128ms        11.17%     194.555ms     194.555us       2.74 Gb           0 b          1000
          aten::max_pool2d_with_indices        10.69%     186.283ms        10.69%     186.283ms     186.283us       2.06 Gb       2.06 Gb          1000
                 Optimizer.step#SGD.step         4.59%      79.953ms         5.81%     101.246ms     101.246us       5.35 Kb      -2.33 Kb          1000
              aten::threshold_backward         3.84%      66.827ms         3.84%      66.827ms      66.827us       2.74 Gb       2.74 Gb          1000
     Optimizer.zero_grad#SGD.zero_grad         1.14%      19.834ms         1.14%      19.834ms      19.834us      -5.22 Mb      -5.22 Mb          1000
aten::max_pool2d_with_indices_backward         1.09%      18.951ms         1.47%      25.646ms      25.646us       2.74 Gb       2.74 Gb          1000
                     aten::_log_softmax         0.89%      15.511ms         0.89%      15.511ms      15.511us      54.82 Mb      54.82 Mb          1000
---------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 1.742s
```

Profiling GPU - fit

```
m_cuda = mnist_conv_model(device="cuda")
with torch.autograd.profiler.profile(with_stack=True) as prof_cuda:
    losses = m_cuda.fit(X_train_cuda, y_train_cuda, n=1000)
```

```
print(prof_cuda.key_averages().table(sort_by='self_cpu_time_total', row_limit=10))
```

```
---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                             Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
          Optimizer.step#SGD.step        21.93%      88.394ms        27.56%     111.084ms     111.084us          1000
                cudaLaunchKernel        11.37%      45.807ms        11.37%      45.807ms       2.082us         21998
Optimizer.zero_grad#SGD.zero_grad         5.11%      20.578ms         5.11%      20.578ms      20.578us          1000
                        aten::mm         3.90%      15.712ms         4.89%      19.704ms       9.852us          2000
                    aten::addmm         3.59%      14.477ms         5.50%      22.178ms      22.178us          1000
       aten::convolution_backward         3.59%      14.457ms         7.48%      30.162ms      30.162us          1000
          aten::cudnn_convolution         3.14%      12.655ms         4.15%      16.715ms      16.715us          1000
                        aten::sum         3.09%      12.450ms         4.07%      16.417ms       8.208us          2000
                    aten::fill_          1.85%       7.469ms         3.51%      14.144ms       4.715us          3000
            cudaStreamSynchronize         1.85%       7.440ms         1.85%       7.440ms       7.440us          1000
---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 403.037ms
```

CIFAR10

homepage

Loading the data

```
import torchvision

training_data = torchvision.datasets.CIFAR10(
    root="/data",
    train=True,
    download=True,
    transform=torchvision.transforms.ToTensor()
)

test_data = torchvision.datasets.CIFAR10(
    root="/data",
    train=False,
    download=True,
    transform=torchvision.transforms.ToTensor()
)
```

Downloads data to “/data/cifar-10-batches-py” which is ~178M on disk.

CIFAR10 data

```
1 training_data.classes
```

```
['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

```
1 training_data.data.shape
```

```
(50000, 32, 32, 3)
```

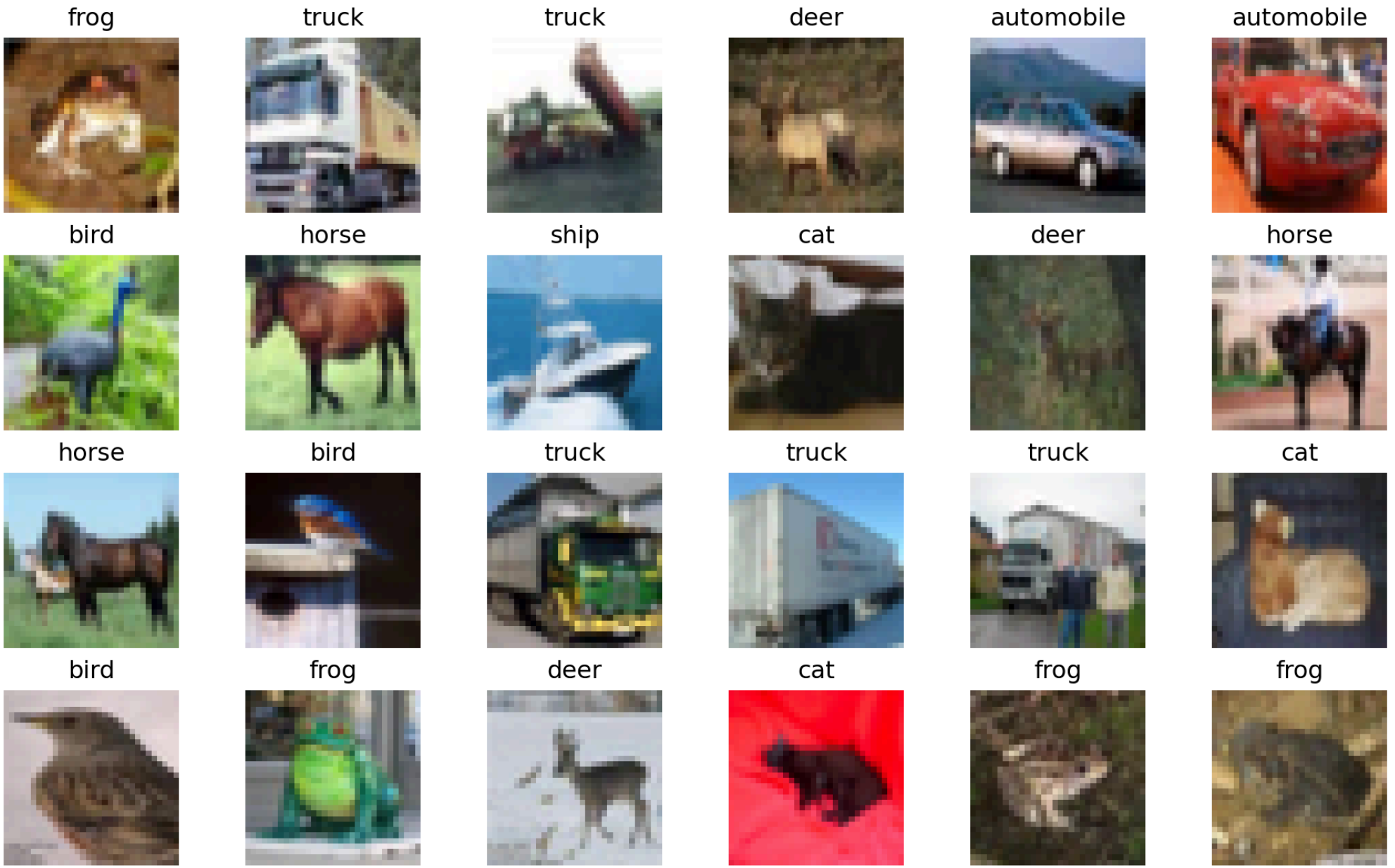
```
1 test_data.data.shape
```

```
(10000, 32, 32, 3)
```

```
1 training_data[0]
```

```
(tensor([[[0.2314, 0.1686, 0.1961, 0.2667,
          0.3843, 0.4667, 0.5451, 0.5686,
          0.5843, 0.5843, 0.5137, 0.4902,
          0.5569, 0.5647, 0.5373, 0.5059,
          0.5373, 0.5255, 0.4863, 0.5451,
          0.5451, 0.5216, 0.5333, 0.5451,
          0.5961, 0.6392, 0.6588, 0.6235,
          0.6196, 0.6196, 0.5961, 0.5804],
         [0.0627, 0.0000, 0.0706, 0.2000,
          0.3451, 0.4706, 0.5020, 0.4980,
          0.4941, 0.4549, 0.4157, 0.3961,
          0.4118, 0.4431, 0.4275, 0.4392,
          0.4667, 0.4275, 0.4118, 0.4902,
          0.4980, 0.4784, 0.5137, 0.4863,
          0.4745, 0.5137, 0.5176, 0.5216,
          0.5216, 0.4824, 0.4667, 0.4784],
         [0.0980, 0.0627, 0.1922, 0.3255,
```

Example data



Data Loaders

Torch handles large datasets and minibatches through the use of the `DataLoader` class,

```
training_loader = torch.utils.data.DataLoader(
    training_data,
    batch_size=100,
    shuffle=True,
    num_workers=4,
    pin_memory=True
)

test_loader = torch.utils.data.DataLoader(
    test_data,
    batch_size=100,
    shuffle=True,
    num_workers=4,
    pin_memory=True
)
```

Loader as generator

The resulting `DataLoader` is iterable and yields the features and targets for each batch,

```
1 training_loader
```

```
<torch.utils.data.dataloader.DataLoader object at 0x7fcfedabb4d0>
```

```
1 X, y = next(iter(training_loader))
2 X.shape
```

```
torch.Size([100, 3, 32, 32])
```

```
1 y.shape
```

```
torch.Size([100])
```

Custom Datasets

In this case we got our data (`training_data` and `test_data`) directly from torchvision which gave us a `dataset` object to use with our `DataLoader`. If we do not have a `Dataset` object then we need to create a custom class for our data telling torch how to load it.

You class must define the methods: `__init__()`, `__len__()`, and `__get_item__()`.

```
class data(torch.utils.data.Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

mnist_train = data(X_train, y_train)
```

Custom loader

```
mnist_loader = torch.utils.data.DataLoader(
    mnist_train,
    batch_size=1000,
    shuffle=True
)

it = iter(mnist_loader)
```

```
X, y = next(it)
X.shape
```

```
torch.Size([1000, 64])
```

```
y.shape
```

```
torch.Size([1000])
```

```
X, y = next(it)
X.shape
```

```
torch.Size([437, 64])
```

```
y.shape
```

```
torch.Size([437])
```

CIFAR CNN

```
class cifar_conv_model(torch.nn.Module):
    def __init__(self, device):
        super().__init__()
        self.device = torch.device(device)
        self.epoch = 0
        self.model = torch.nn.Sequential(
            torch.nn.Conv2d(3, 6, kernel_size=5),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2, 2),
            torch.nn.Conv2d(6, 16, kernel_size=5),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2, 2),
            torch.nn.Flatten(),
            torch.nn.Linear(16 * 5 * 5, 120),
            torch.nn.ReLU(),
            torch.nn.Linear(120, 84),
            torch.nn.ReLU(),
            torch.nn.Linear(84, 10)
        ).to(device=self.device)

    def forward(self, X):
        return self.model(X)

    def fit(self, loader, epochs=10, n_report=250, lr=0.001):
        opt = torch.optim.SGD(self.parameters(), lr=lr, momentum=0.9)

```

CNN Performance - CPU (1 step)

```
X, y = next(iter(training_loader))

m_cpu = cifar_conv_model(device="cpu")
tmp = m_cpu(X)

with torch.autograd.profiler.profile(with_stack=True) as prof_cpu:
    tmp = m_cpu(X)
```

```
print(prof_cpu.key_averages().table(sort_by='self_cpu_time_total', row_limit=10))
```

```
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                           Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                    aten::addmm        71.56%       2.956ms        72.02%       2.975ms     991.781us             3
     aten::mkldnn_convolution        17.14%     707.940us        17.80%     735.292us     367.646us             2
aten::max_pool2d_with_indices         4.64%     191.532us         4.64%     191.532us      95.766us             2
               aten::clamp_min         2.93%     121.099us         2.93%     121.099us      30.275us             4
            aten::convolution         0.72%      29.726us        18.87%     779.376us     389.688us             2
                   aten::relu         0.39%      16.261us         3.32%     137.360us      34.340us             4
                  aten::empty         0.39%      16.040us         0.39%      16.040us       4.010us             4
                  aten::copy_         0.35%      14.387us         0.35%      14.387us       4.796us             3
           aten::_convolution         0.35%      14.358us        18.15%     749.650us     374.825us             2
            aten::as_strided_         0.23%       9.599us         0.23%       9.599us       4.799us             2
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 4.131ms
```

CNN Performance - GPU (1 step)

```
m_cuda = cifar_conv_model(device="cuda")
Xc, yc = X.to(device="cuda"), y.to(device="cuda")
tmp = m_cuda(Xc)

with torch.autograd.profiler.profile(with_stack=True) as prof_cuda:
    tmp = m_cuda(Xc)
```

```
print(prof_cuda.key_averages().table(sort_by='self_cpu_time_total', row_limit=10))
```

```
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                           Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
      aten::cudnn_convolution        57.65%     698.424us        60.15%     728.650us     364.325us             2
                    cudaMalloc        15.09%     182.856us        15.09%     182.856us     182.856us             1
                   aten::addmm         5.66%      68.620us         7.35%      88.988us      29.663us             3
              cudaLaunchKernel         5.20%      62.948us         5.20%      62.948us       3.934us            16
               aten::clamp_min         2.47%      29.876us        18.57%     224.946us      56.236us             4
                    aten::add_         2.21%      26.780us         2.80%      33.973us      16.986us             2
             aten::convolution         2.20%      26.642us        67.38%     816.296us     408.148us             2
aten::max_pool2d_with_indices         1.90%      22.973us         2.34%      28.323us      14.161us             2
            aten::_convolution         1.63%      19.757us        65.18%     789.654us     394.827us             2
                    aten::relu         1.35%      16.401us        19.92%     241.347us      60.337us             4
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 1.211ms
```

CNN Performance - CPU (1 epoch)

```
m_cpu = cifar_conv_model(device="cpu")

with torch.autograd.profiler.profile(with_stack=True) as prof_cpu:
    m_cpu.fit(loader=training_loader, epochs=1, n_report=501)
```

```
print(prof_cpu.key_averages().table(sort_by='self_cpu_time_total', row_limit=10))
```

```
-------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                                 Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
-------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                           aten::mm        21.26%     410.672ms        21.29%     411.375ms     137.125us          3000
          aten::convolution_backward        16.97%     327.782ms        17.40%     336.087ms     336.087us          1000
                      aten::addmm        12.83%     247.920ms        13.14%     253.859ms     169.239us          1500
            aten::mkldnn_convolution         9.74%     188.087ms        10.12%     195.513ms     195.513us          1000
enumerate(DataLoader)#_MultiProcessingDataLoaderIter...         8.47%     163.597ms         8.50%     164.116ms     327.577us           501
              Optimizer.step#SGD.step         5.12%      98.973ms         7.47%     144.393ms     288.786us           500
       aten::max_pool2d_with_indices         4.97%      95.998ms         4.97%      96.062ms      96.062us          1000
            aten::threshold_backward         4.14%      79.915ms         4.14%      79.970ms      39.985us          2000
                     aten::clamp_min         2.52%      48.692ms         2.52%      48.718ms      24.359us          2000
aten::max_pool2d_with_indices_backward         1.56%      30.183ms         2.94%      56.851ms      56.851us          1000
-------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 1.932s
```

CNN Performance - GPU (1 epoch)

```
m_cuda = cifar_conv_model(device="cuda")

with torch.autograd.profiler.profile(with_stack=True) as prof_cuda:
    m_cuda.fit(loader=training_loader, epochs=1, n_report=501)
```

```
print(prof_cuda.key_averages().table(sort_by='self_cpu_time_total', row_limit=10))
```

```
---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                             Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
enumerate(DataLoader)#_MultiProcessingDataLoaderIter...        46.81%     548.984ms        46.87%     549.669ms       1.097ms           501
            cudaStreamSynchronize         8.90%     104.432ms         8.91%     104.446ms      69.631us          1500
          Optimizer.step#SGD.step         6.77%      79.444ms         8.34%      97.846ms     195.691us           500
                cudaLaunchKernel         6.07%      71.196ms         6.07%      71.201ms       2.739us         25998
       aten::convolution_backward         2.60%      30.487ms         6.20%      72.736ms      72.736us          1000
                        aten::mm         2.10%      24.631ms         2.76%      32.337ms      10.779us          3000
                     aten::addmm         1.78%      20.865ms         2.53%      29.632ms      19.755us          1500
Optimizer.zero_grad#SGD.zero_grad         1.72%      20.189ms         1.72%      20.190ms      40.379us           500
                       aten::sum         1.55%      18.127ms         2.10%      24.685ms       9.874us          2500
          aten::cudnn_convolution         1.46%      17.153ms         1.99%      23.287ms      23.287us          1000
---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 1.173s
```

Loaders & Accuracy

```python
def accuracy(model, loader, device):
    total, correct = 0, 0
    with torch.no_grad():
        for X, y in loader:
            X, y = X.to(device=device), y.to(device=device)
            pred = model(X)
            # the class with the highest energy is what we choose as prediction
            val, idx = torch.max(pred, 1)
            total += pred.size(0)
            correct += (idx == y).sum().item()

    return correct / total
```

Model fitting

```
m = cifar_conv_model("cuda")
m.fit(training_loader, epochs=10, n_report=500, lr=0.01)
```

```
## [Epoch 1, Minibatch  500] loss: 2.098
## [Epoch 2, Minibatch  500] loss: 1.692
## [Epoch 3, Minibatch  500] loss: 1.482
## [Epoch 4, Minibatch  500] loss: 1.374
## [Epoch 5, Minibatch  500] loss: 1.292
## [Epoch 6, Minibatch  500] loss: 1.226
## [Epoch 7, Minibatch  500] loss: 1.173
## [Epoch 8, Minibatch  500] loss: 1.117
## [Epoch 9, Minibatch  500] loss: 1.071
## [Epoch 10, Minibatch  500] loss: 1.035
```

```
accuracy(m, training_loader, "cuda")
```

```
## 0.63444
```

```
accuracy(m, test_loader, "cuda")
```

```
## 0.572
```

More epochs

If we fit again, Torch continues with the existing model,

```
m.fit(training_loader, epochs=10, n_report=500)
```

```
## [Epoch 11, Minibatch  500] loss: 0.885
## [Epoch 12, Minibatch  500] loss: 0.853
## [Epoch 13, Minibatch  500] loss: 0.839
## [Epoch 14, Minibatch  500] loss: 0.828
## [Epoch 15, Minibatch  500] loss: 0.817
## [Epoch 16, Minibatch  500] loss: 0.806
## [Epoch 17, Minibatch  500] loss: 0.798
## [Epoch 18, Minibatch  500] loss: 0.787
## [Epoch 19, Minibatch  500] loss: 0.780
## [Epoch 20, Minibatch  500] loss: 0.773
```

```
accuracy(m, training_loader, "cuda")
```

```
## 0.73914
```

```
accuracy(m, test_loader, "cuda")
```

```
## 0.624
```

More epochs (again)

```
m.fit(training_loader, epochs=10, n_report=500)
```

```
## [Epoch 21, Minibatch  500] loss: 0.764
## [Epoch 22, Minibatch  500] loss: 0.756
## [Epoch 23, Minibatch  500] loss: 0.748
## [Epoch 24, Minibatch  500] loss: 0.739
## [Epoch 25, Minibatch  500] loss: 0.733
## [Epoch 26, Minibatch  500] loss: 0.726
## [Epoch 27, Minibatch  500] loss: 0.718
## [Epoch 28, Minibatch  500] loss: 0.710
## [Epoch 29, Minibatch  500] loss: 0.702
## [Epoch 30, Minibatch  500] loss: 0.698
```

```
accuracy(m, training_loader, "cuda")
```

```
## 0.76438
```

```
accuracy(m, test_loader, "cuda")
```

```
## 0.6217
```

The VGG16 model

```
class VGG16(torch.nn.Module):
    def make_layers(self):
        cfg = [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512, 'M']
        layers = []
        in_channels = 3
        for x in cfg:
            if x == 'M':
                layers += [torch.nn.MaxPool2d(kernel_size=2, stride=2)]
            else:
                layers += [torch.nn.Conv2d(in_channels, x, kernel_size=3, padding=1),
                           torch.nn.BatchNorm2d(x),
                           torch.nn.ReLU(inplace=True)]
                in_channels = x
        layers += [
            torch.nn.AvgPool2d(kernel_size=1, stride=1),
            torch.nn.Flatten(),
            torch.nn.Linear(512,10)
        ]

        return torch.nn.Sequential(*layers).to(self.device)

    def __init__(self, device):
        super().__init__()
        self.device = torch.device(device)
        self.model = self.make_layers()

```

Model

```
1  VGG16("cpu").model
```

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (5): ReLU(inplace=True)
  (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (9): ReLU(inplace=True)
  (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (12): ReLU(inplace=True)
  (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (16): ReLU(inplace=True)
  (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

VGG16 performance - CPU

```
X, y = next(iter(training_loader))
m_cpu = VGG16(device="cpu")
tmp = m_cpu(X)

with torch.autograd.profiler.profile(with_stack=True) as prof_cpu:
    tmp = m_cpu(X)
```

```
print(prof_cpu.key_averages().table(sort_by='self_cpu_time_total', row_limit=10))
```

```
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                           Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
     aten::mkldnn_convolution        82.23%      50.624ms        82.46%      50.766ms       3.905ms            13
      aten::native_batch_norm         9.81%       6.039ms         9.96%       6.133ms     471.805us            13
aten::max_pool2d_with_indices         5.85%       3.602ms         5.85%       3.602ms     720.367us             5
             aten::clamp_min_         0.69%     424.702us         0.69%     424.702us      32.669us            13
                  aten::empty         0.28%     171.053us         0.28%     171.053us       1.316us           130
           aten::convolution          0.18%     113.285us        82.78%      50.961ms       3.920ms            13
                 aten::relu_          0.16%      97.607us         0.85%     522.309us      40.178us            13
                 aten::addmm          0.15%      90.031us         0.16%      99.709us      99.709us             1
                  aten::add_          0.15%      89.901us         0.15%      89.901us       6.915us            13
          aten::_convolution          0.13%      81.593us        82.59%      50.848ms       3.911ms            13
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 61.564ms
```

VGG16 performance - GPU

```
m_cuda = VGG16(device="cuda")
Xc, yc = X.to(device="cuda"), y.to(device="cuda")
tmp = m_cuda(Xc)

with torch.autograd.profiler.profile(with_stack=True) as prof_cuda:
    tmp = m_cuda(Xc)
```

```
print(prof_cuda.key_averages().table(sort_by='self_cpu_time_total', row_limit=10))
```

```
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                           Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                     cudaMalloc        39.07%       1.450ms        39.07%       1.450ms      80.534us            18
        aten::cudnn_convolution        26.17%     971.002us        58.81%       2.182ms     167.836us            13
              cudaLaunchKernel         6.34%     235.060us         6.34%     235.060us       2.374us            99
         aten::cudnn_batch_norm         5.38%     199.459us        14.95%     554.510us      42.655us            13
                    aten::empty         3.87%     143.460us         9.91%     367.644us       5.570us            66
                     aten::add_         3.67%     136.061us         4.97%     184.519us       7.097us            26
             aten::_convolution         2.33%      86.376us        64.89%       2.408ms     185.211us            13
              aten::convolution         2.14%      79.554us        67.04%       2.487ms     191.331us            13
                     aten::relu_         1.75%      64.744us         3.81%     141.499us      10.885us            13
aten::max_pool2d_with_indices         1.53%      56.598us         6.35%     235.597us      47.119us             5
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 3.710ms
```

VGG16 performance - Apple M1 GPU (mps)

```
m_mps = VGG16(device="mps")
Xm, ym = X.to(device="mps"), y.to(device="mps")

with torch.autograd.profiler.profile(with_stack=True) as prof_mps:
    tmp = m_mps(Xm)
```

```
print(prof_mps.key_averages().table(sort_by='self_cpu_time_total', row_limit=10))
```

```
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                           Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
         aten::native_batch_norm        35.71%       3.045ms        35.71%       3.045ms     234.231us            13
          aten::_mps_convolution        19.67%       1.677ms        19.88%       1.695ms     130.385us            13
    aten::_batch_norm_impl_index        11.92%       1.016ms        36.02%       3.071ms     236.231us            13
                     aten::relu_        11.29%     963.000us        11.29%     963.000us      74.077us            13
                      aten::add_        10.40%     887.000us        10.44%     890.000us      68.462us            13
-------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 8.526ms
```

Fitting w/ `lr = 0.01`

```
m = VGG16(device="cuda")
fit(m, training_loader, epochs=10, n_report=500, lr=0.01)
```

```
## [Epoch 1, Minibatch  500] loss: 1.345
## [Epoch 2, Minibatch  500] loss: 0.790
## [Epoch 3, Minibatch  500] loss: 0.577
## [Epoch 4, Minibatch  500] loss: 0.445
## [Epoch 5, Minibatch  500] loss: 0.350
## [Epoch 6, Minibatch  500] loss: 0.274
## [Epoch 7, Minibatch  500] loss: 0.215
## [Epoch 8, Minibatch  500] loss: 0.167
## [Epoch 9, Minibatch  500] loss: 0.127
## [Epoch 10, Minibatch  500] loss: 0.103
```

```
accuracy(model=m, loader=training_loader, device="cuda")
```

```
## 0.97008
```

```
accuracy(model=m, loader=test_loader, device="cuda")
```

```
## 0.8318
```

Fitting w/ `lr = 0.001`

```
m = VGG16(device="cuda")
fit(m, training_loader, epochs=10, n_report=500, lr=0.001)
```

```
## [Epoch 1, Minibatch  500] loss: 1.279
## [Epoch 2, Minibatch  500] loss: 0.827
## [Epoch 3, Minibatch  500] loss: 0.599
## [Epoch 4, Minibatch  500] loss: 0.428
## [Epoch 5, Minibatch  500] loss: 0.303
## [Epoch 6, Minibatch  500] loss: 0.210
## [Epoch 7, Minibatch  500] loss: 0.144
## [Epoch 8, Minibatch  500] loss: 0.108
## [Epoch 9, Minibatch  500] loss: 0.088
## [Epoch 10, Minibatch  500] loss: 0.063
```

```
accuracy(model=m, loader=training_loader, device="cuda")
```

```
## 0.9815
```

```
accuracy(model=m, loader=test_loader, device="cuda")
```

```
## 0.7816
```

Report

```python
from sklearn.metrics import classification_report

def report(model, loader, device):
    y_true, y_pred = [], []
    with torch.no_grad():
        for X, y in loader:
            X = X.to(device=device)
            y_true.append( y.cpu().numpy() )
            y_pred.append( model(X).max(1)[1].cpu().numpy() )

    y_true = np.concatenate(y_true)
    y_pred = np.concatenate(y_pred)

    return classification_report(y_true, y_pred, target_names=loader.dataset.classes)
```

```
print(report(model=m, loader=test_loader, device="cuda"))
```

```
##                precision    recall  f1-score   support
##
##     airplane       0.82      0.88      0.85      1000
##   automobile       0.95      0.89      0.92      1000
##         bird       0.85      0.70      0.77      1000
##          cat       0.68      0.74      0.71      1000
##         deer       0.84      0.83      0.83      1000
##          dog       0.81      0.73      0.77      1000
##         frog       0.83      0.92      0.87      1000
##        horse       0.87      0.87      0.87      1000
##         ship       0.89      0.92      0.90      1000
##        truck       0.86      0.93      0.89      1000
##
##     accuracy                           0.84     10000
##    macro avg       0.84      0.84      0.84     10000
## weighted avg       0.84      0.84      0.84     10000
```

Some “state-of-the-art” models

Hugging Face

This is an online community and platform for sharing machine learning models (architectures and weights), data, and related artifacts. They also maintain a number of packages and related training materials that help with building, training, and deploying ML models.

Some notable resources,

- `transformers` - APIs and tools to easily download and train state-of-the-art (pretrained) transformer based models
- `diffusers` - provides pretrained vision and audio diffusion models, and serves as a modular toolbox for inference and training
- `timm` - a library containing SOTA computer vision models, layers, utilities, optimizers, schedulers, data-loaders, augmentations, and training/evaluation scripts

Stable Diffusion

```
import torch
from diffusers import StableDiffusionPipeline, DPMSolverMultistepScheduler

model_id = "/data/stable-diffusion-2-1"

pipe = StableDiffusionPipeline.from_pretrained(model_id, torch_dtype=torch.float16)
```

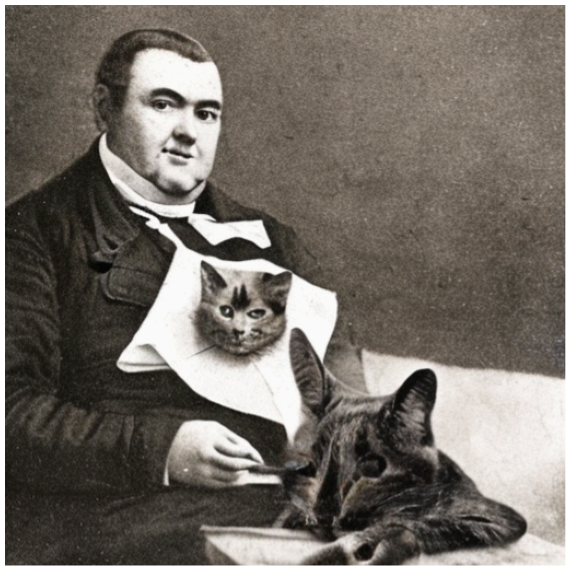
```
pipe.scheduler = DPMSolverMultistepScheduler.from_config(pipe.scheduler.config)
pipe.enable_model_cpu_offload()
```

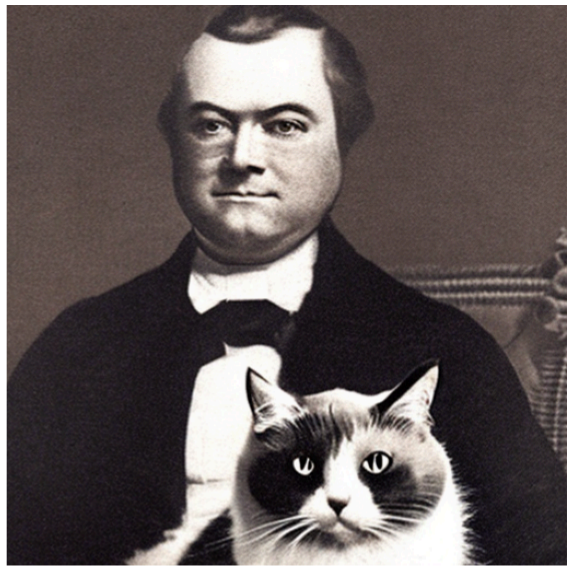
```
prompt = "a picture of thomas bayes with a cat on his lap"
```

```
prompt = "a picture of thomas bayes with a cat on his lap"
generator = [torch.Generator(device="cuda").manual_seed(i) for i in range(6)]
fit = pipe(prompt, generator=generator, num_inference_steps=20, num_images_per_prompt=6)
```

```
fit.images
```

```
[<PIL.Image.Image image mode=RGB size=512x512 at 0x7F3E885A2BA0>, <PIL.Image.Image image mode=RGB size=512x512 at
   0x7F3E885A16D0>, <PIL.Image.Image image mode=RGB size=512x512 at 0x7F3E885A0E90>, <PIL.Image.Image image mode=RGB
   size=512x512 at 0x7F3E885A28D0>, <PIL.Image.Image image mode=RGB size=512x512 at 0x7F3E885A1EB0>, <PIL.Image.Image
   image mode=RGB size=512x512 at 0x7F3E885A1D30>]
```

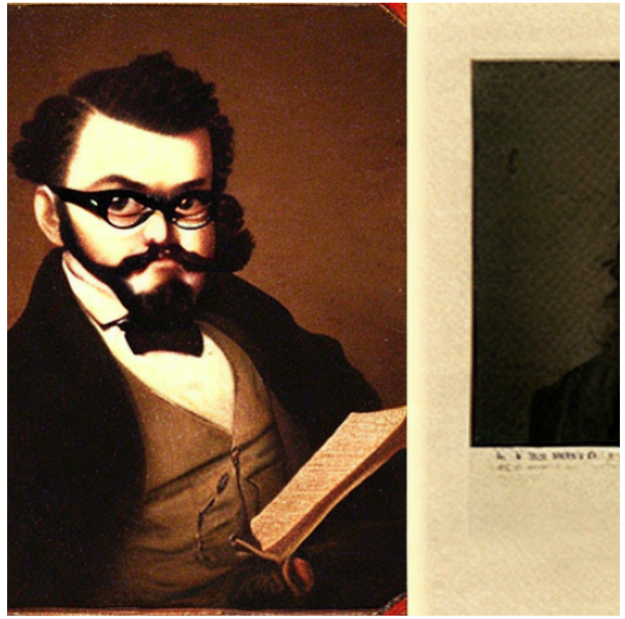





Customizing prompts

```
prompt = "a picture of thomas bayes with a cat on his lap"
prompts = [
  prompt + t for t in
  ["in the style of a japanese wood block print",
   "as a hipster with facial hair and glasses",
   "as a simpsons character, cartoon, yellow",
   "in the style of a vincent van gogh painting",
   "in the style of a picasso painting",
   "with flowery wall paper"
  ]
]
```

```
generator = [torch.Generator(device="cuda").manual_seed(i) for i in range(6)]
fit = pipe(prompts, generator=generator, num_inference_steps=20, num_images_per_prompt=1)
```



Increasing inference steps

```
generator = [torch.Generator(device="cuda").manual_seed(i) for i in range(6)]
fit = pipe(prompts, generator=generator, num_inference_steps=50, num_images_per_prompt=1)
```

A more current model

This model is larger than the available GPU memory - so we adjust the weight types to make it fit.

```python
from diffusers import BitsAndBytesConfig, SD3Transformer2DModel
from diffusers import StableDiffusion3Pipeline
import torch

model_id = "/data/stable-diffusion-3.5-medium"

nf4_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)
model_nf4 = SD3Transformer2DModel.from_pretrained(
    model_id,
    subfolder="transformer",
    quantization_config=nf4_config,
    torch_dtype=torch.bfloat16
)

pipe = StableDiffusion3Pipeline.from_pretrained(
    model_id,
    transformer=model_nf4,
    torch_dtype=torch.bfloat16
)
```

```
Loading pipeline components...:   0%|          | 0/9 [00:00<?, ?it/s]
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s] [A
Loading checkpoint shards:  50%|#####     | 1/2 [00:00<00:00,  2.01it/s] [A
Loading checkpoint shards: 100%|##########| 2/2 [00:00<00:00,  2.15it/s] [ALoading checkpoint shards: 100%|##########|
```

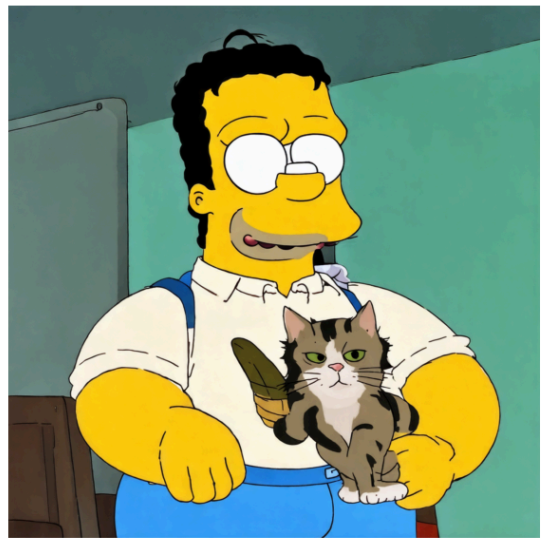
```
   2/2 [00:00<00:00,  2.13it/s]
Loading pipeline components...:  44%|####4     | 4/9 [00:01<00:01,  3.80it/s]Loading pipeline components...:  56%|
   #####5    | 5/9 [00:01<00:00,  4.48it/s]Loading pipeline components...: 100%|##########| 9/9 [00:01<00:00,  7.45it/
   s]Loading pipeline components...: 100%|##########| 9/9 [00:01<00:00,  6.15it/s]
```

```
1 pipe.enable_model_cpu_offload()
```

Images

```
generator = [torch.Generator(device="cuda").manual_seed(i) for i in range(6)]
fit = pipe(prompts, generator=generator, num_inference_steps=30, num_images_per_prompt=1)
```






LLM - Qwen2.5-3B

```python
from transformers import pipeline

generator = pipeline('text-generation', model='Qwen/Qwen2.5-3B-Instruct')
prompt = "Can you write me a short bed time story about Thomas Bayes and his pet cat? Limit the story to no more than

result = generator(
    prompt, max_length=500, num_return_sequences=1,
    truncation=True
)

print( result[0]['generated_text'] )
```

```
Can you write me a short bed time story about Thomas Bayes and his pet cat? Limit the story to no more than three
paragraphs.

In a quiet corner of a small, cozy house lived Thomas Bayes, a man known for his clever mind but often found lost in
thought. His favorite place was a little study where he spent most of his days pondering the mysteries of probability.
One day, as he sat by the window watching the clouds drift by, he noticed a curious feline peeking through the curtains.
Intrigued, he let the cat inside, naming it Bayes after himself, much to the cat's delight. The cat, with its sleek black
fur and piercing green eyes, became Bayes' constant companion. They would sit together on Bayes' lap, discussing theories
over soft purrs, until the night grew dark and Bayes retired to his study to continue his work, always with Bayes curled
up at his feet, a silent witness to his intellectual journey. As the moonlight filtered through the windows, Bayes would
drift off to sleep, dreaming of the future of statistics, with Bayes, his loyal and curious friend, by his side.
```

Qwen is an opensource LLM built by Alibaba