In [1]:

```
! python --version
```

```
Python 3.6.9
```

In [2]:

```
! pip freeze | grep -P '(torch|numpy)'
```

```
numpy==1.19.2
pytorch-lightning==1.3.7.post0
torch==1.8.1+cpu
torchmetrics==0.4.0
torchtext==0.3.1
torchvision==0.9.1+cpu
```

In [3]:

```
! lscpu
```

```
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              8
On-line CPU(s) list: 0-7
Thread(s) per core:  2
Core(s) per socket:  4
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               142
Model name:          Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
Stepping:            12
CPU MHz:             2381.660
CPU max MHz:         4600.0000
CPU min MHz:         400.0000
BogoMIPS:            3999.93
Virtualization:      VT-x
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            8192K
NUMA node0 CPU(s):   0-7
Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtr
r pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs
bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulq
dq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm p
cid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave a
vx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb invpcid_sin
gle ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi flexpriority
ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid
mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsav
es dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp md_
clear flush_l1d arch_capabilities
```

In [4]:

```
! nvcc --version
```

```
/bin/bash: nvcc: command not found
```

In [5]:

```
! nvidia-smi --query-gpu=name,driver_version --format=csv
```

```
/bin/bash: nvidia-smi: command not found
```

In [6]:

```
import torch
torch.set_printoptions(precision=11)
```

In [7]:

```
print(torch.__config__.show())
```

```
PyTorch built with:
  - GCC 7.3
  - C++ Version: 201402
  - Intel(R) Math Kernel Library Version 2020.0.0 Product Build 2019
1122 for Intel(R) 64 architecture applications
  - Intel(R) MKL-DNN v1.7.0 (Git Hash 7aed236906b1f7a05c0917e5257a1a
f05e9ff683)
  - OpenMP 201511 (a.k.a. OpenMP 4.5)
  - NNPACK is enabled
  - CPU capability usage: AVX2
  - Build settings: BLAS_INFO=mkl, BUILD_TYPE=Release, CXX_COMPILER
=/opt/rh/devtoolset-7/root/usr/bin/c++, CXX_FLAGS= -Wno-deprecated -
fvisibility-inlines-hidden -DUSE_PTHREADPOOL -fopenmp -DNDEBUG -DUSE
_FBGEMM -DUSE_QNNPACK -DUSE_PYTORCH_QNNPACK -DUSE_XNNPACK -O2 -fPIC
-Wno-narrowing -Wall -Wextra -Werror=return-type -Wno-missing-field-
initializers -Wno-type-limits -Wno-array-bounds -Wno-unknown-pragmas
-Wno-sign-compare -Wno-unused-parameter -Wno-unused-variable -Wno-un
used-function -Wno-unused-result -Wno-unused-local-typedefs -Wno-str
ict-overflow -Wno-strict-aliasing -Wno-error=deprecated-declarations
-Wno-stringop-overflow -Wno-psabi -Wno-error=pedantic -Wno-error=red
undant-decls -Wno-error=old-style-cast -fdiagnostics-color=always -f
aligned-new -Wno-unused-but-set-variable -Wno-maybe-uninitialized -f
no-math-errno -fno-trapping-math -Werror=format -Wno-stringop-overfl
ow, LAPACK_INFO=mkl, PERF_WITH_AVX=1, PERF_WITH_AVX2=1, PERF_WITH_AV
X512=1, TORCH_VERSION=1.8.1, USE_CUDA=0, USE_CUDNN=OFF, USE_EXCEPTIO
N_PTR=1, USE_GFLAGS=OFF, USE_GLOG=OFF, USE_MKL=ON, USE_MKLDNN=ON, US
E_MPI=OFF, USE_NCCL=OFF, USE_NNPACK=ON, USE_OPENMP=ON,
```

In [8]:

```
print('Cuda', torch.cuda.is_available())
```

```
Cuda False
```

In [9]:

```
def set_random_seed(seed_value: int, use_cuda: bool = False):
    import torch
    import numpy as np
    import random
    np.random.seed(seed_value)  # cpu vars
    torch.manual_seed(seed_value)  # cpu  vars
    random.seed(seed_value)  # Python
    torch.use_deterministic_algorithms(True)
    if use_cuda:
        torch.cuda.manual_seed(seed_value)
        torch.cuda.manual_seed_all(seed_value)  # gpu vars
        torch.backends.cudnn.deterministic = True  # needed
        torch.backends.cudnn.benchmark = True


def _run_module(layer_norm, dropout, x, use_cuda, train, seed):
    if use_cuda:
        device = torch.device('cuda:0')
    else:
        device = torch.device('cpu')
    set_random_seed(seed, use_cuda=use_cuda)
    layers = []
    if layer_norm:
        layers.append(torch.nn.LayerNorm(normalized_shape=(x.shape[-1],), eps=1e
-12, elementwise_affine=True))
    if dropout:
        layers.append(torch.nn.Dropout(0.5, inplace=False))

    if not layers:
        raise ValueError('set `layer_norm` and/or `dropout` to True')

    model = torch.nn.Sequential(*layers)
    model.to(device)
    if train:
        model.zero_grad()
        model.train()
    else:
        model.eval()
    x = x.clone().to(device)
    if train:
        out = model(x)
    else:
        with torch.no_grad():
            out = model(x)
    return out


def run_layer_norm_module(seed):
    set_random_seed(seed, use_cuda=False)
    x = torch.rand(3, 3)
    return {
        'x': x,
        'cpu_t': _run_module(layer_norm=True, dropout=False, x=x, use_cuda=False
, train=True, seed=seed),
        'cpu_e': _run_module(layer_norm=True, dropout=False, x=x, use_cuda=False
, train=False, seed=seed),
        # 'gpu_t': _run_module(layer_norm=True, dropout=False, x=x, use_cuda=Tru
e, train=True, seed=seed),
        # 'gpu_e': _run_module(layer_norm=True, dropout=False, x=x, use_cuda=Tru
```

```
e, train=False, seed=seed)
    }



def run_dropout_module(seed):
    set_random_seed(seed, use_cuda=False)
    x = torch.rand(4, 256, 256)
    return {
        'x': x,
        'cpu_t': _run_module(layer_norm=False, dropout=True, x=x, use_cuda=False
, train=True, seed=seed),
        # 'gpu_t': _run_module(layer_norm=False, dropout=True, x=x, use_cuda=Tru
e, train=True, seed=seed),
    }
```

In [10]:

```
runs = [run_dropout_module(seed=42) for i in range(5)]

for i in range(1, len(runs)):
    for k in runs[0]:
        assert torch.equal(runs[0][k], runs[i][k]), (i, k)

for k in runs[0]:
    if k != 'x':
        print(k, (runs[0][k].cpu() == 0.0).sum(axis=-1).tolist())
        print('=' * 100)
```

```
cpu_t [[119, 120, 120, 127, 126, 116, 131, 124, 125, 120, 123, 128,
130, 132, 134, 124, 133, 135, 130, 128, 119, 138, 125, 123, 132, 13
4, 137, 142, 114, 120, 131, 136, 124, 125, 155, 118, 144, 125, 129,
135, 124, 126, 134, 121, 121, 136, 127, 130, 124, 117, 106, 124, 13
2, 143, 124, 128, 136, 120, 119, 123, 116, 127, 138, 117, 124, 137,
128, 119, 123, 124, 138, 128, 118, 127, 134, 136, 133, 123, 123, 13
1, 119, 123, 136, 128, 123, 138, 127, 125, 133, 116, 125, 135, 130,
117, 127, 131, 125, 137, 123, 137, 137, 128, 122, 126, 132, 139, 14
1, 127, 112, 145, 148, 143, 130, 118, 129, 143, 136, 122, 121, 124,
140, 127, 116, 125, 126, 133, 124, 132, 138, 136, 135, 126, 134, 13
2, 123, 135, 119, 131, 121, 126, 128, 125, 121, 131, 126, 131, 126,
142, 130, 119, 135, 128, 118, 136, 132, 137, 135, 119, 121, 129, 13
2, 124, 114, 137, 138, 117, 128, 127, 139, 132, 126, 134, 129, 131,
127, 118, 123, 130, 133, 136, 138, 140, 128, 119, 130, 134, 129, 13
1, 122, 124, 138, 125, 127, 123, 127, 124, 142, 122, 132, 126, 117,
122, 123, 136, 118, 130, 122, 135, 125, 132, 119, 129, 137, 122, 13
0, 126, 144, 130, 143, 122, 119, 131, 123, 122, 117, 121, 127, 120,
137, 127, 132, 127, 134, 124, 139, 142, 122, 129, 116, 120, 121, 13
5, 123, 125, 129, 131, 126, 131, 121, 120, 121, 134, 120, 128, 152,
114], [126, 136, 128, 128, 126, 135, 120, 130, 121, 119, 124, 138, 1
22, 122, 134, 127, 127, 130, 118, 128, 124, 128, 125, 137, 134, 127,
125, 117, 121, 128, 117, 121, 116, 116, 144, 131, 128, 126, 124, 11
7, 143, 131, 135, 145, 133, 122, 122, 129, 127, 126, 119, 122, 120,
127, 132, 134, 132, 119, 132, 132, 141, 128, 132, 133, 132, 118, 13
2, 129, 121, 136, 131, 127, 135, 120, 128, 124, 137, 127, 121, 130,
133, 135, 129, 130, 131, 131, 118, 129, 119, 127, 140, 134, 127, 13
6, 138, 125, 125, 133, 121, 111, 124, 120, 133, 133, 134, 136, 143,
130, 126, 122, 137, 139, 147, 126, 138, 134, 117, 130, 122, 134, 13
8, 120, 128, 117, 132, 120, 138, 131, 136, 127, 124, 131, 124, 112,
142, 118, 121, 129, 117, 116, 136, 137, 120, 134, 138, 132, 128, 12
3, 137, 132, 123, 131, 122, 117, 125, 108, 123, 127, 136, 125, 126,
130, 123, 118, 135, 110, 135, 122, 114, 136, 125, 128, 124, 128, 13
1, 129, 119, 130, 124, 134, 132, 121, 133, 138, 129, 116, 116, 125,
142, 107, 110, 135, 130, 128, 119, 142, 119, 124, 137, 129, 132, 11
5, 117, 125, 118, 141, 131, 117, 138, 125, 120, 130, 126, 125, 123,
136, 137, 127, 122, 135, 138, 114, 129, 142, 119, 129, 113, 135, 13
3, 135, 130, 137, 141, 143, 131, 128, 137, 137, 122, 123, 123, 141,
125, 141, 109, 136, 133, 116, 112, 126, 128, 115, 125, 132, 149, 12
8], [125, 138, 126, 128, 136, 128, 127, 126, 124, 129, 133, 138, 12
3, 128, 130, 124, 133, 123, 125, 128, 128, 139, 137, 136, 130, 130,
132, 134, 118, 108, 125, 126, 125, 122, 130, 133, 137, 135, 133, 12
6, 132, 134, 134, 116, 135, 129, 135, 129, 120, 131, 120, 132, 121,
108, 132, 128, 130, 125, 137, 117, 128, 139, 123, 138, 128, 111, 12
2, 112, 133, 124, 126, 137, 130, 143, 121, 130, 124, 125, 130, 116,
126, 129, 134, 129, 138, 121, 133, 124, 135, 126, 129, 132, 131, 11
8, 133, 135, 118, 132, 112, 119, 125, 119, 140, 129, 132, 138, 120,
130, 129, 125, 123, 111, 129, 121, 129, 126, 126, 142, 130, 132, 13
3, 113, 121, 133, 136, 125, 126, 127, 140, 137, 120, 135, 134, 133,
131, 125, 120, 136, 129, 121, 126, 134, 136, 130, 132, 132, 125, 12
7, 116, 140, 133, 125, 130, 135, 134, 119, 120, 115, 131, 127, 121,
133, 119, 124, 129, 122, 129, 114, 121, 117, 119, 119, 119, 133, 15
0, 131, 127, 117, 117, 139, 130, 131, 129, 128, 128, 120, 130, 137,
132, 129, 114, 134, 127, 116, 119, 128, 112, 126, 110, 122, 133, 12
4, 132, 141, 128, 113, 126, 120, 124, 135, 129, 117, 129, 125, 138,
125, 129, 129, 124, 136, 127, 135, 128, 128, 141, 128, 133, 128, 11
7, 134, 116, 132, 130, 122, 127, 127, 137, 132, 137, 115, 119, 125,
123, 134, 116, 120, 127, 133, 122, 125, 123, 117, 139, 134, 120, 13
1], [131, 118, 138, 120, 148, 111, 122, 120, 123, 130, 145, 119, 12
5, 141, 124, 126, 116, 125, 135, 130, 114, 134, 129, 132, 115, 124,
129, 127, 116, 122, 146, 136, 115, 129, 135, 123, 119, 132, 149, 14
3, 129, 120, 119, 122, 119, 128, 134, 136, 133, 123, 136, 117, 127,
```

```
126, 135, 122, 130, 129, 122, 139, 143, 133, 125, 128, 125, 138, 11
4, 136, 131, 129, 134, 120, 122, 127, 118, 121, 124, 141, 126, 126,
130, 131, 121, 116, 146, 130, 138, 121, 139, 115, 119, 125, 134, 12
0, 139, 144, 126, 128, 121, 123, 129, 128, 137, 126, 128, 129, 131,
107, 136, 135, 118, 126, 134, 129, 126, 142, 137, 125, 128, 143, 12
5, 135, 133, 120, 112, 123, 141, 126, 123, 140, 130, 125, 120, 138,
126, 123, 133, 143, 123, 118, 134, 128, 144, 125, 127, 115, 139, 12
1, 133, 128, 119, 131, 126, 124, 149, 138, 132, 137, 136, 118, 132,
138, 136, 123, 134, 118, 119, 136, 124, 127, 123, 124, 123, 121, 13
6, 132, 130, 114, 139, 132, 130, 123, 130, 132, 105, 136, 126, 125,
125, 124, 127, 115, 129, 131, 136, 120, 128, 131, 136, 118, 132, 12
0, 130, 116, 136, 129, 129, 132, 129, 134, 113, 140, 117, 134, 118,
133, 117, 148, 134, 129, 129, 128, 139, 128, 126, 126, 121, 130, 13
4, 133, 124, 145, 137, 126, 120, 134, 129, 124, 129, 147, 132, 108,
127, 138, 132, 113, 113, 140, 119, 118, 127, 130, 117, 118, 133, 12
6]]
================================================================
================================
```

In [11]:

```python
runs = [run_layer_norm_module(seed=42) for i in range(5)]

for i in range(1, len(runs)):
    for k in runs[0]:
        assert torch.equal(runs[0][k], runs[i][k]), (i, k)

runs[0]
```

Out[11]:

```
{'x': tensor([[0.88226926327, 0.91500395536, 0.38286375999],
         [0.95930564404, 0.39044821262, 0.60089534521],
         [0.25657248497, 0.79364132881, 0.94077146053]]),
 'cpu_t': tensor([[ 0.63882547617,  0.77325701714, -1.41208255291],
         [ 1.31617474556, -1.10615468025, -0.21002103388],
         [-1.38439559937,  0.44202369452,  0.94237166643]],
        grad_fn=<NativeLayerNormBackward>),
 'cpu_e': tensor([[ 0.63882547617,  0.77325701714, -1.41208255291],
         [ 1.31617474556, -1.10615468025, -0.21002103388],
         [-1.38439559937,  0.44202369452,  0.94237166643]])}
```