PyTorch Cheat Sheet

Using PyTorch 1.2, torchaudio 0.3, torchtext 0.4, and torchvision 0.4.

General PyTorch and model I/O

```
# loading PyTorch
import torch

# cuda
import torch.cuda as tCuda # various functions and settings
torch.backends.cudnn.deterministic = True # deterministic ML?
torch.backends.cudnn.benchmark = False  # deterministic ML?
torch.cuda.is_available # check if cuda is is_available
tensor.cuda() # moving tensor to gpu
tensor.cpu() # moving tensor to cpu
tensor.to(device) # copy densor to device xyz
torch.device('cuda') # or 'cuda0', 'cuda1' if multiple devices
torch.device('cpu') # default

# static computation graph/C++ export preparation
torch.jit.trace()
from torch.jit import script, trace
@script

# load and save a model
torch.save(model, 'model_file')
model = torch.load('model_file')
model.eval() # set to inference
torch.save(model.state_dict(), 'model_file') # only state dict
model = ModelCalss()
model.load_state_dict(torch.load('model_file')

# save to onnx
torch.onnx.export
torch.onnx.export_to_pretty_string

# load onnx model
import onnx
model = onnx.load('model.onnx')
# check model
onnx.checker.check_model(model)
```

Pre-trained models and domain-specific utils

Audio

```
import torchaudio
# load and save audio
stream, sample_rate = torchaudio.load('file')
torchaudio.save('file', stream, sample_rate)
# 16 bit wav files only
stream, sample_rate=torchaudio.load_wav('file')

# datasets (can be used with torch.utils.data.DataLoader)
import torchaudio.datasets as aDatasets
aDatasets.YESNO('folder_for_storage', download=True)
aDatasets.VCTK('folder_for_storage', download=True)


# transforms
import torchaudio.transforms as aTransforms
aTransforms.AmplitudeToDB
aTransforms.MelScale
aTransforms.MelSpectrogram
aTransforms.MFCC
aTransforms.MuLawEncoding
aTransforms.MuLawDecoding
aTransforms.Resample
aTransforms.Spectogram

# kaldi support
import torchaudio.compliance.kaldi as aKaldi
import torchaudio.kaldi_io as aKaldiIO
aKaldi.spectogram
aKaldi.fbank
aKaldi.mfcc
aKaldi.resample_waveform
```

```
aKaldiIO.read_vec_int_ark
aKaldiIO.read_vec_flt_scp
aKaldiIO.read_vec_flt_ark
aKaldiIO.read_mat_scp
aKaldiIO.read_mat_ark

# functional/direct function access
import torchaudio.functional as aFunctional

# sox effects/passing data between Python and C++
import torchaudio.sox_effects as aSox_effects
```

Text

```
import torchtext
# various data-related function and classes
import torchtext.data as tData
tData.Batch
tData.Dataset
tData.Example
tData.TabularDataset
tData.RawField
tData.Field
tData.ReversibleField
tData.SubwordField
tData.NestedField
tData.Iterator
tData.BucketIterator
tData.BPTTIterator
tData.Pipeline # similar to vTransform and sklearn's pipeline
tData.batch # function
tData.pool # function


# datasets
import torchtext.datasets as tDatasets
# sentiment analysis
tDatasets.SST
tDatasets.IMDb
tDatasets.TextClassificationDataset # subclass of all datasets below
tDatasets.AG_NEWS
tDatasets.SogouNews
tDatasets.DBpedia
tDatasets.YelpReviewPolarity
tDatasets.YelpReviewFull
tDatasets.YahooAnswers
tDatasets.AmazonReviewPolarity
tDatasets.AmazonReviewFull

# question classification
tDatasets.TREC

# entailment
tDatasets.SNLI
tDatasets.MultiNLI

# language modeling
tDatasets.WikiText2
tDatasets.WikiText103
tDatasets.PennTreebank

# machine translation
tDatasets.TranslationDataset # subclass
tDatasets.Multi30k
tDatasets.IWSLT
tDatasets.WMT14

# sequence tagging
tDatasets.SequenceTaggingDataset # subclass
tDatasets.UDPOS
tDatasets.CoNLL2000Chunking

# question answering
tDatasets.BABI20

# vocabulary and pre-trained embeddings
import torchtext.vocab as tVocab
tVocab.Vocab # create a vocabulary
tVocab.SubwordVocab # create subvocabulary
tVocab.Vectors # word vectors
tVocab.GloVe # GloVe embeddings
tVocab.FastText # FastText embeddings
tVocab.CharNGram # character n-gram
```

Vision

```
import torchvision
# datasets
import torchvision.datasets as vDatasets
vDatasets.MNIST
vDatasets.FashionMNIST
vDatasets.KMNIST
vDatasets.EMNIST
vDatasets.QMNIST
vDatasets.FakeData # randomly generated images
vDatasets.COCOCaptions
vDatasets.COCODetection
vDatasets.LSUN
vDatasets.ImageFolder # data loader for a certain image folder structure
vDatasets.DatasetFolder # data loader for a certain folder structure
vDatasets.ImageNet
vDatasets.CIFAR
vDatasets.STL10
vDatasets.SVHN
vDatasets.PhotoTour
vDatasets.SBU
vDatasets.Flickr
vDatasets.VOC
vDatasets.Cityscapes
vDatasets.SBD
vDatasets.USPS
vDatasets.Kinetics400
vDatasets.HMDB51
vDatasets.UCF101

# video IO
import torchvision.io as vIO
vIO.read_video('file', start_pts, end_pts)
vIO.write_video('file', video, fps, video_codec)
torchvision.utils.save_image(image,'file')

# pretrained models/model architectures
import torchvision.models as vModels
# models can be constructed with random weights ()
# or pretrained (pretrained=True)
# classification
vModels.alexnet(pretrained=True)
vModels.densenet121()
vModels.densenet161()
vModels.densenet169()
vModels.densenet201()
vModels.googlenet()
vModels.inception_v3()
vModels.mnasnet0_5()
vModels.mnasnet0_75()
vModels.mnasnet1_0()
vModels.mnasnet1_3()
vModels.mobilenet_v2()
vModels.resnet18()
vModels.resnet34()
vModels.resnet50()
vModels.resnet50_32x4d()
vModels.resnet101()
vModels.resnet101_32x8d()
vModels.resnet152()
vModels.wide_resnet50_2()
vModels.wide_resnet101_2()
vModels.shufflenet_v2_x0_5()
vModels.shufflenet_v2_x1_0()
vModels.shufflenet_v2_x1_5()
vModels.shufflenet_v2_x2_0()
vModels.squeezenet1_0()
vModels.squeezenet1_1()
vModels.vgg11()
vModels.vgg11_bn()
vModels.vgg13()
vModels.vgg13_bn()
vModels.vgg16()
vModels.vgg16_bn()
vModels.vgg19()
vModels.vgg19_bn()

# semantic segmentation
vModels.segmentation.fcn_resnet50()
vModels.segmentation.fcn_resnet101()
vModels.segmentation.deeplabv3_resnet50()
```

```
vModels.segmentation.deeplabv3_resnet101()

# object and/or keypoint detection, instance segmentation
vModels.detection.fasterrcnn_resnet50_fpn()
vModels.detection.maskrcnn_resnet50_fpn()
vModels.detection.keypointrcnn_resnet50_fpn()

# video classification
vModels.video.r3d_18()
vModels.video.mc3_18()
vModels.video.r2plus1d_18()

# transforms
import torchvision.transforms as vTransforms
vTransforms.Compose(transforms) # chaining transforms
vTransforms.Lambda(someLambdaFunction)

# transforms on PIL images
vTransforms.CenterCrop(height,width)
vTransforms.ColorJitter(brightness=0, contrast=0,
                        saturation=0, hue=0)
vTransforms.FiveCrop
vTransforms.Grayscale
vTransforms.Pad
vTransforms.RandomAffine(degrees, translate=None,
                         scale=None, shear=None,
                         resample=False, fillcolor=0)
vTransforms.RandomApply(transforms, p=0.5)
vTransforms.RandomChoice(transforms)
vTransforms.RandomCrop
vTransforms.RandomGrayscale
vTransforms.RandomHorizontalFlip
vTransforms.RandomOrder
vTransforms.RandomPerspective
vTransforms.RandomResizedCrop
vTransforms.RandomRotation
vTransforms.RandomSizedCrop
vTransforms.RandomVerticalFlip
vTransforms.Resize
vTransforms.Scale
vTransforms.TenCrop

# transforms on torch tensors
vTransforms.LinearTransformation
vTransforms.Normalize
vTransforms.RandomErasing

# conversion
vTransforms.ToPILImage
vTransforms.ToTensor

# direct access to transform functions
import torchvision.transforms.functional as vTransformsF

# operators for computer vision
# (not supported by TorchScript yet)
import torchvision.ops as vOps
vOps.nms # non-maximum suppression (NMS)
vOps.roi_align # <=> vOps.ROIALIGN
vOps.roi_pool # <=> vOps.ROIPOOL
```

Data loader

```
# classes and functions to represent datasets
from torch.utils.data import Dataset, Dataloader
```

Neural network

```
import torch.nn as nn
```

Activation functions

```
nn.AdaptiveLogSoftmaxWithLoss
nn.CELU
nn.EL
nn.Hardshrink
nn.Hardtanh
nn.LeakyReLU
nn.LogSigmoid
nn.LogSoftmax
nn.MultiheadAttention
```

```
nn.PReLU
nn.ReLU
nn.ReLU6
nn.RReLU(lower,upper) # sampled from uniform distribution
nn.SELU
nn.Sigmoid
nn.Softmax
nn.Softmax2d
nn.Softmin
nn.Softplus
nn.Softshrink
nn.Softsign
nn.Tanh
nn.Tanhshrink
nn.Threshols
```

Loss function

```
nn.BCELoss
nn.BCEWithLogitsLoss
nn.CosineEmbeddingLoss
nn.CrossEntropyLoss
nn.CTCLoss
nn.HingeEmbeddingLoss
nn.KLDivLoss
nn.L1Loss
nn.MarginRankingLoss
nn.MSELoss
nn.MultiLabelSoftMarginLoss
nn.MultiMarginLoss
nn.NLLLoss
nn.PoissonNLLLoss
nn.SmoothL1Loss
nn.SoftMarginLoss
nn.TripletMarginLoss
```

Optimizer

```
import torch.optim as optim
# general useage
scheduler = optim.Optimizer(....)
scheduler.step() # step-wise

optim.lr_scheduler.Scheduler

# optimizers
optim.Optimizer # general optimizer classes
optim.Adadelta
optim.Adagrad
optim.Adam
optim.AdamW # adam with decoupled weight decay regularization
optim.Adamax
optim.ASGD # averged stochastic gradient descent
optim.LBFGS
optim.RMSprop
optim.Rprop
optim.SGD
optim.SparseAdam # for sparse tensors

# learning rate
optim.lr_scheduler
optim.lr_scheduler.LambdaLR
optim.lr_scheduler.StepLR
optim.lr_scheduler.MultiStepLR
optim.lr_scheduler.ExponentialLR
optim.lr_scheduler.CosineAnnealingLR
optim.lr_scheduler.ReduceLROnPlateau
optim.lr_scheduler.CyclicLR
```

Pre-defined layers/deep learning

```
# containers
nn.Module{ ,List,Dict}
nn.Parameter{List,Dict}
nn.Sequential

# linear layers
nn.Linear
nn.Bilinear
nn.Indentity

# dropout layers
nn.AlphaDropout
nn.Dropout{ ,2d,3d}
```

```
# convolutional layers
nn.Conv{1,2,3}d
nn.ConvTranspose{1,2,3}d
nn.Fold
nn.Unfold

# pooling
nn.AdaptiveAvgPool{1,2,3}d
nn.AdaptiveMaxPool{1,2,3}d
nn.AvgPool{1,2,3}d
nn.MaxPool{1,2,3}d
nn.MaxUnpool{1,2,3}d

# recurrent layers
nn.GRU
nn.LSTM
nn.RNN

# padding layers
nn.ReflectionPad{1,2}d
nn.ReplicationPad{1,2,3}d
nn.ConstantPad{1,2,3}d

# normalization layers
nn.BatchNorm{1,2,3}d
nn.InstanceNorm{1,2,3}d

# transformer layers
nn.Transformer
nn.TransformerEncoder
nn.TransformerDecoder
nn.TransformerEncoderLayer
nn.TransformerDecoderLayer
```

Computational graph

```
# various functions and classes to use and manipulate
# automaic differentiation and the computational graph
import torch.autograd as autograd
```

Functional

```
import torch.nn.functional as F
# direct function access and not via classes (torch.nn) ???
```

NumPy-like functions

Loading PyTorch and tensor basics

```
# loading PyTorch
import torch

# defining a tensor
torch.tensor((values))

# define data type
torch.tensor((values), dtype=torch.int16)

# converting a NumPy array to a PyTorch tensor
torch.from_numpy(numpyArray)

# create a tensor of zeros
torch.zeros((shape))
torch.zeros_like(other_tensor)

# create a tensor of ones
torch.ones((shape))
torch.ones_like(other_tensor)

# create an idenity matrix
torch.eye(numberOfRows)

# create tensor with same values
torch.full((shape), value)
torch.full_like(other_tensor,value)

# create an empty tensor
torch.empty((shape))
torch.empty_like(other_tensor)

# create sequences
torch.arange(startNumber, endNumber, stepSize)
```

```
torch.linspace(startNumber, endNumber, stepSize)
torch.logspace(startNUmber, endNumber, stepSize)

# concatenate tensors
torch.cat((tensors), axis)

# split tensors into sub-tensors
torch.split(tensor, splitSize)

# (un)squeeze tensor
torch.squeeze(tensor, dimension)
torch.unsqueeze(tensor, dim)

# reshape tensor
torch.reshape(tensor, shape)

# transpose tensor
torch.t(tensor) # 1D and 2D tensors
torch.transpose(tensor, dim0, dim1)
```

Random numbers

```
# set seed
torch.manual_seed(seed)

# generate a tensor with random numbers
# of interval [0,1)
torch.rand(size)
torch.rand_like(other_tensor)

# generate a tensor with random integer numbers
# of interval [lowerInt,higherInt]
torch.randint(lowerInt,
              higherInt,
              (tensor_shape))
torch.randint_like(other_tensor,
                   lowerInt,
                   higherInt)

# generate a tensor of random numbers drawn
# from a normal distribution (mean=0, var=1)
torch.randn((size))
torch.randn_like(other_tensor)

# random permuation of integers
# range [0,n-1)
torch.randperm()
```

Math (element-wise)

```
# basic operations
torch.abs(tensor)
torch.add(tensor, tensor2) # or tensor+scalar
torch.div(tensor, tensor2) # or tensor/scalar
torch.mult(tensor,tensor2) # or tensor*scalar
torch.sub(tensor, tensor2) # or tensor-scalar
torch.ceil(tensor)
torch.floor(tensor)
torch.remainder(tensor, devisor) #or torch.fmod()
torch.sqrt(tensor)

# trigonometric functions
torch.acos(tensor)
torch.asin(tensor)
torch.atan(tensor)
torch.atan2(tensor)
torch.cos(tensor)
torch.cosh(tensor)
torch.sin(tensor)
torch.sinh(tensor)
torch.tan(tensor)
torch.tanh(tensor)

# exponentials and logarithms
torch.exp(tensor)
torch.expm1(tensor) # exp(input-1)
torch.log(tensor)
torch.log10(tensor)
torch.log1p(tensor) # log(1+input)
torch.log2(tensor)

# other
torch.erfc(tensor) # error function
torch.erfinv(tensor) # inverse error function
```

```
torch.round(tensor) # round to full integer
torch.power(tensor, power)
```

Math (not element-wise)

```
torch.argmax(tensor)
torch.argmin(tensor)
torch.max(tensor)
torch.min(tensor)
torch.mean(tensor)
torch.median(tensor)
torch.norm(tensor, norm)
torch.prod(tensor) # product of all elements
torch.std(tensor)
torch.sum(tensor)
torch.unique(tensor)
torch.var(tensor)
torch.cross(tensor1,tensor2)
torch.cartesian_prod(tensor1, tensor2, ...)
torch.einsum(equation,tensor)
torch.tensordot(tensor1,tensor2)
torch.cholesky(tensor)
torch.cholesky_torch(tensor)
torch.dot(tensor1, tensor2)
torch.eig(tensor)
torch.inverse(tensor)
torch.det(tensor)
```



```
torch.pinverse(tensor) # pseudo-inverse
```

Other

```
torch.isinf(tensor)
torch.sort(tensor)
torch.fft(tensor, signal_dim)
torch.ifft(tensor, signal_dim)
torch.rfft(tensor, signal_dim)
torch.rifft(tensor, signal_dim)
torch.stft(tensor, n_fft)
torch.bincount(tensor)
torch.diagonal(tensor)
torch.flatten(tensor, start_dim)
torch.rot90(tensor)
torch.histc(tensor)
torch.trace(tensor)
torch.svd(tensor)
```

PyTorch C++

(aka libtorch)

```
// PyTorch header file(s)
#import <torch/script.h>

torch::jit::script::Module module;
```