Pytorch Cheat Sheet

Shivanshu Gupta

Imports

General

```
import torch                                          # root package
from torch.utils.data import Dataset, Dataloader      # dataset representation and
                                                      # loading
```

Neural Network API

```
import torch.autograd as autograd      # computation graph
from torch import Tensor               # tensor node in the computation graph
import torch.nn as nn                  # neural networks
import torch.nn.functional as F        # layers, activations and more
import torch.optim as optim            # optimizers e.g. SGD, ADAM, etc.
from torch.jit import script, trace    # hybrid frontend decorator and tracing jit
```

See autograd, nn, functional and optim

Torchscript and JIT

```
torch.jit.trace()    # takes your module or function and an example
                     # data input, and traces the computational steps
                     # that the data encounters as it progresses through the model
@script              # decorator used to indicate data-dependent
                     # control flow within the code being traced
```

See Torchscript

ONNX

```
torch.onnx.export(model, dummy data, xxxx.proto)    # exports an ONNX formatted
                                                    # model using a trained model,
                                                    # dummy data and the desired
                                                    # file name
model = onnx.load("alexnet.proto")                  # load an ONNX model
onnx.checker.check_model(model)                     # check that the model
                                                    # IR is well formed
onnx.helper.printable_graph(model.graph)            # print a human readable
                                                    # representation of the graph
```

See onnx

Vision

```
from torchvision import datasets, models, transforms    # vision datasets,
                                                        # architectures &
                                                        # transforms
import torchvision.transforms as transforms             # composable transforms
```

See torchvision

Distributed Training

```
import torch.distributed as dist          # distributed communication
from multiprocessing import Process       # memory sharing processes
```

See distributed and multiprocessing

Tensors

Creation

```
torch.randn(*size)          # tensor with independent N(0,1) entries
torch.[ones|zeros](*size)   # tensor with all 1's [or 0's]
torch.Tensor(L)             # create tensor from [nested] list or ndarray L
x.clone()                   # clone of x
with torch.no_grad():       # code wrap that stops autograd from tracking
                            # tensor history
requires_grad=True          # arg, when set to True, tracks computation
                            # history for future derivative calculations
```

See tensor

Dimensionality

```
x.size()                          # return tuple-like object of dimensions
torch.cat(tensor_seq, dim=0)      # concatenates tensors along dim
x.view(a,b,...)                   # reshapes x into size (a,b,...)
x.view(-1,a)                      # reshapes x into size (b,a) for some b
x.transpose(a,b)                  # swaps dimensions a and b
x.permute(*dims)                  # permutes dimensions
x.unsqueeze(dim)                  # tensor with added axis
x.unsqueeze(dim=2)                # (a,b,c) tensor -> (a,b,1,c) tensor
```

See tensor

Algebra

```
A.mm(B)     # matrix multiplication
A.mv(x)     # matrix-vector multiplication
x.t()       # matrix transpose
```

See math operations

GPU Usage

```
torch.cuda.is_available                 # check for cuda
x.cuda()                                # move x's data from
                                        # CPU to GPU and return new object

x.cpu()                                 # move x's data from GPU to CPU
                                        # and return new object

if not args.disable_cuda \
    and torch.cuda.is_available():      # device agnostic code
    args.device = torch.device('cuda')  # and modularity
else:                                   #
    args.device = torch.device('cpu')   #

net.to(device)                          # recursively convert their
                                        # parameters and buffers to
                                        # device specific tensors

mytensor.to(device)                     # copy your tensors to a device
                                        # (gpu, cpu)
```

See cuda

Deep Learning

Layers

```
nn.Linear(m,n)                          # fully connected layer from
                                        # m to n units

nn.ConvXd(m,n,s)                        # X dimensional conv layer from
                                        # m to n channels where X???1,2,3
                                        # and the kernel size is s

nn.MaxPoolXd(s)                         # X dimension pooling layer
                                        # (notation as above)

nn.BatchNorm                            # batch norm layer
nn.RNN/LSTM/GRU                         # recurrent layers
nn.Dropout(p=0.5, inplace=False)        # dropout layer for any dimensional
                                        # input
nn.Dropout2d(p=0.5, inplace=False)      # 2-dimensional channel-wise dropout
nn.Embedding(num_embeddings, embedding_dim) # (tensor-wise) mapping from
                                        # indices to embedding vectors
```

See nn

Loss Functions

```
nn.X                        # where X is BCELoss, CrossEntropyLoss,
                            # L1Loss, MSELoss, NLLLoss, SoftMarginLoss,
                            # MultiLabelSoftMarginLoss, CosineEmbeddingLoss,
                            # KLDivLoss, MarginRankingLoss, HingeEmbeddingLoss
                            # or CosineEmbeddingLoss
```

See loss functions

Activation Functions

```
nn.X                        # where X is ReLU, ReLU6, ELU, SELU, PReLU, LeakyReLU,
                            # Threshold, HardTanh, Sigmoid, Tanh,
                            # LogSigmoid, Softplus, SoftShrink,
                            # Softsign, TanhShrink, Softmin, Softmax,
                            # Softmax2d or LogSoftmax
```

See activation functions

Optimizers

```
opt = optim.x(model.parameters(), ...)  # create optimizer
opt.step()                              # update weights
optim.X                                 # where X is SGD, Adadelta, Adagrad, Adam,
                                        # SparseAdam, Adamax, ASGD,
                                        # LBFGS, RMSProp or Rprop
```

See optimizers

Learning rate scheduling

```
scheduler = optim.X(optimizer,...)  # create lr scheduler
scheduler.step()                    # update lr at start of epoch
optim.lr_scheduler.X                # where X is LambdaLR, StepLR, MultiStepLR,
                                    # ExponentialLR or ReduceLROnPLateau
```

See learning rate scheduler

Data Utilities

Datasets

```
Dataset                 # abstract class representing dataset
TensorDataset           # labelled dataset in the form of tensors
Concat Dataset          # concatenation of Datasets
```

See datasets

Dataloaders and DataSamplers

```
DataLoader(dataset, batch_size=1, ...)  # loads data batches agnostic
                                        # of structure of individual data points

sampler.Sampler(dataset,...)            # abstract class dealing with
                                        # ways to sample from dataset

sampler.XSampler where ...              # Sequential, Random, Subset,
                                        # WeightedRandom or Distributed
```

See dataloader