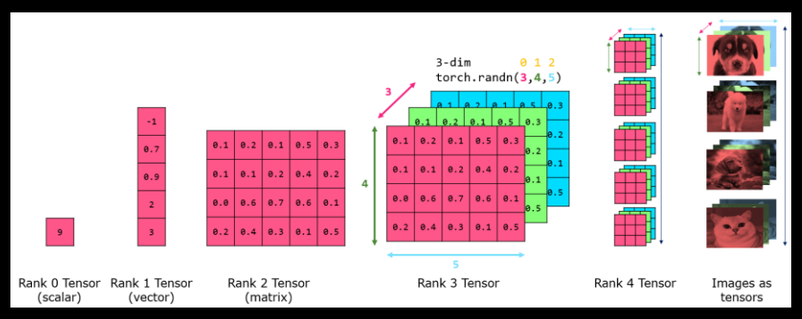
PyTorch CHEAT SHEET

Imports

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




```
# N-dim of tensor = number of square brackets [ on the left side
# [batch_size, hight, width, color_channels] NHWC # images alternative 1
# [batch_size, color_channels, hight, width] NCHW # images alternative 2
```

Tensors Creation

```
x = torch.tensor(data,dtype,device) # tensor with no autograd history
x = torch.arange(start,end,step)    # seqeuence
x = torch.randn(*size)              # tensor of random numbers from normal distribution N(0,1)
x = torch.[ones|zeros](*size)       # tensor with all 1's [or 0's]
x = torch.tensor(L)                 # tensor from [nested] list or ndarray L
x = torch.from_numpy(numpy_arr)     # tensor from a NumPy array
y = 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
```

Dimensionality

```
x.size()                            # return tuple-like object of dimensions
x = torch.cat(tensors,dim=0)        # concatenates tensors along dim WITHOUT changing the dim of
                                    # tensors
y = torch.stack(tensors,dim=0)      # stacks a sequence of tensors along a NEW dimension
y = x.view(a,b,...)                 # reshapes x into size (a,b,...)
y = x.view(-1,a)                    # reshapes x into size (b,a) for some b
y = x.transpose(a,b)                # swaps dimensions a and b
y = x.permute(*dims)                # Returns a view of the original input with its dimensions
                                    # permuted (rearranged) to dims
y = x.unsqueeze(dim)                # tensor with added axis
y = x.unsqueeze(dim=2)              # (a,b,c) tensor -> (a,b,1,c) tensor
y = x.squeeze()                     # removes all dimensions of size 1 (a,1,b,1) -> (a,b)
y = x.squeeze(dim=1)                # removes specified dimension of size 1 (a,1,b,1) -> (a,b,1)
y = x.reshape(shape)                # Reshapes input to shape (if compatible)
```

Math

```
# Algebra
ret = A * B         # element-wise multiplication
ret = A.mm(B)       # matrix multiplication / dot product
ret = A.mv(x)       # matrix-vector multiplication
x = x.t()           # matrix transpose

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)
```

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
```

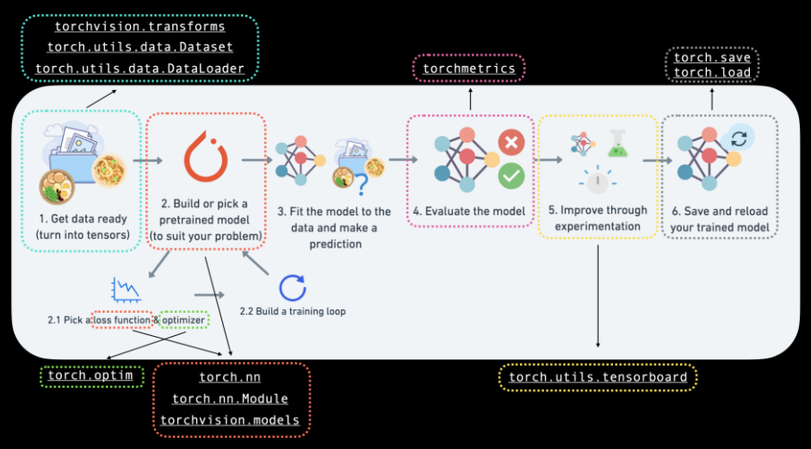
ONNX

```
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
```

PyTorch Workflow



Data Utilities

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

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, SubsetRandom,
                                    # WeightedRandom, Batch, Distributed
```

Vision

```
# Base computer vision library
import torchvision

# Other components of TorchVision (premade datasets, pretrained models and image transforms)
from torchvision import datasets, models, transforms
```

Text

```
# Base text and natural language processing library
import torchtext

# Other components of TorchText (premade datasets, pretrained models and text transforms)
from torchtext import datasets, models, transforms
```

Audio and Speech

```
# Base audio and speech processing library
import torchaudio

# Other components of TorchAudio (premade datasets, pretrained models and text transforms)
from torchaudio import datasets, models, transforms
```

Recommendation systems

```
# Base recommendation system library
import torchrec

# Other components of TorchRec
from torchrec import datasets, models
```

GPU Usage

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

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

if torch.cuda.is_available():           # Setup device-agnostic code
    device = "cuda"                     # NVIDIA GPU
elif torch.backends.mps_is_available():
    device = "mps"                      # Apple GPU
else:
    device = "cpu"



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

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

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. gradient descent, ADAM, etc.
from torch.jit import script, trace     # hybrid frontend decorator and tracing jit
```

Deep Learning

```
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.BatchNormXd                          # 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
```

Model Building

```
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super().__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out

model = NeuralNet(input_size, hidden_size, num_classes).to(device)
```

Loss Functions

```
nn.X                        # where X is L1Loss, MSELoss, CrossEntropyLoss
                            # CTCLoss, NLLLoss, PoissonNLLLoss,
                            # KLDivLoss, BCELoss, BCEWithLogitsLoss,
                            # MarginRankingLoss, HingeEmbeddingLoss,
                            # MultiLabelMarginLoss, SmoothL1Loss,
                            # SoftMarginLoss, MultiLabelSoftMarginLoss,
                            # CosineEmbeddingLoss, MultiMarginLoss,
                            # or TripletMarginLoss
```

Activation Functions

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

Optimizers

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

Learning rate scheduling

```
scheduler = optim.X(optimizer,...)      # create lr scheduler
scheduler.step()                        # update lr after optimizer updates weights
optim.lr_scheduler.X                    # where X is LambdaLR, MultiplicativeLR,
                                        # StepLR, MultiStepLR, ExponentialLR,
                                        # CosineAnnealingLR, ReduceLROnPlateau, CyclicLR,
                                        # OneCycleLR, CosineAnnealingWarmRestarts,
```

Distributed Training

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

Ahmad Alismail, linkedin.com/in/ahmadalismail1

Last updated: 11/2022

PyTorch Workflow: Daniel Bourke https://github.com/mrdbourke/pytorch-deep-learning