PyTorch-101: Introduction to PyTorch

What is PyTorch?

1. PyTorch is a Python library which facilitate building deep learning models easily.
2. Used for applications such as computer vision and natural language processing, etc.
3. Developed by Facebook AI Research (FAIR) lab.
4. PyTorch emphasizes flexibility and allows deep learning models to be expressed in idiomatic Python, which makes it a strong contender to replace TensorFlow as first choice.
5. It allows you to work with NumPy-like arrays on GPUs.
6. PyTorch supports dynamic computation graph.
7. Helps in automatically computing gradients.

Why PyTorch?

1. TensorFlow has a steep learning curve, and is not “so Pythonic” to easily grasp.
2. Creating a static dynamic graph in TensorFlow beforehand is unnecessary. Instead, PyTorch allows creation of dynamic computation graph on the fly!
3. PyTorch allows automated differentiation and gradient passing. (AutoGrad)
4. It is a replacement for NumPy to use the power of GPUs and other accelerators.

Diving into the library: NN Layers

```
torch.nn.RNN(*args, **kwargs)
torch.nn.LSTM(*args, **kwargs)
torch.nn.GRU(*args, **kwargs)
torch.nn.RNNCell(input_size, hidden_size, bias=True, nonlinearity='tanh')
torch.nn.LSTMCell(input_size, hidden_size, bias=True)
torch.nn.GRUCell(input_size, hidden_size, bias=True)
torch.nn.Linear(in_features, out_features, bias=True)
torch.nn.Bilinear(in1_features, in2_features, out_features, bias=True)
torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)
torch.nn.Conv3d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)
torch.nn.ConvTranspose1d(in_channels, out_channels, kernel_size, stride=1, padding=0, output_padding=0, groups=1,
torch.nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1, padding=0, output_padding=0, groups=1,
torch.nn.ConvTranspose3d(in_channels, out_channels, kernel_size, stride=1, padding=0, output_padding=0, groups=1,
torch.nn.Unfold(kernel_size, dilation=1, padding=0, stride=1)
torch.nn.Fold(output_size, kernel_size, dilation=1, padding=0, stride=1)
```

Diving into the library: Loss Functions

```
torch.nn.L1Loss(size_average=None, reduce=None, reduction='mean') # L1 Loss
torch.nn.MSELoss(size_average=None, reduce=None, reduction='mean') # Mean square error loss
torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean')
torch.nn.CTCLoss(blank=0, reduction='mean') #Connectionist Temporal Classification loss
torch.nn.NLLLoss(weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean') #negative log likelihoo
torch.nn.PoissonNLLLoss(log_input=True, full=False, size_average=None, eps=1e-08, reduce=None, reduction='mean')
torch.nn.KLDivLoss(size_average=None, reduce=None, reduction='mean') # Kullback-Leibler divergence Loss
torch.nn.BCELoss(weight=None, size_average=None, reduce=None, reduction='mean') # Binary Cross Entropy
torch.nn.MarginRankingLoss(margin=0.0, size_average=None, reduce=None, reduction='mean')
```

Diving into the library: Pooling Layers

```
torch.nn.MaxPool1d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)
torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)
torch.nn.MaxPool3d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)
torch.nn.MaxUnpool2d(kernel_size, stride=None, padding=0) # Computes a partial inverse of MaxPool2d
torch.nn.AvgPool2d(kernel_size, stride=None, padding=0, ceil_mode=False, count_include_pad=True)
torch.nn.FractionalMaxPool2d(kernel_size, output_size=None, output_ratio=None, return_indices=False, _random_samples=None)
torch.nn.LPPool2d(norm_type, kernel_size, stride=None, ceil_mode=False) # 2D power-average pooling
torch.nn.AdaptiveMaxPool2d(output_size, return_indices=False)
torch.nn.AdaptiveAvgPool2d(output_size)
```

Diving into the library: Activation Functions

```
torch.nn.ELU(alpha=1.0, inplace=False) #  the element-wise function
torch.nn.Hardshrink(lambd=0.5) #  hard shrinkage function element-wise
torch.nn.LeakyReLU(negative_slope=0.01, inplace=False)
torch.nn.PReLU(num_parameters=1, init=0.25)
torch.nn.ReLU(inplace=False)
torch.nn.RReLU(lower=0.125, upper=0.3333333333333333, inplace=False) # randomized leaky rectified liner unit function
torch.nn.SELU(inplace=False)
torch.nn.CELU(alpha=1.0, inplace=False)
torch.nn.Sigmoid()
torch.nn.Softplus(beta=1, threshold=20)
torch.nn.Softshrink(lambd=0.5)
torch.nn.Tanh()
torch.nn.Tanhshrink()
torch.nn.Threshold(threshold, value, inplace=False)
torch.nn.Softmax(dim=None)
torch.nn.Softmax2d()
```

Diving into the library: Models

```
# model with random weights
import torchvision.models as models
resnet18 = models.resnet18()
alexnet = models.alexnet()
vgg16 = models.vgg16()
squeezenet = models.squeezenet1_0()
densenet = models.densenet161()
inception = models.inception_v3()
googlenet = models.googlenet()
# with pre-trained models
resnet18 = models.resnet18(pretrained=True)
alexnet = models.alexnet(pretrained=True)
squeezenet = models.squeezenet1_0(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
densenet = models.densenet161(pretrained=True)
inception = models.inception_v3(pretrained=True)
googlenet = models.googlenet(pretrained=True)
```

Dynamic Computation Graphs

A graph is created on the fly

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```

PyTorch sub-libraries for special tasks

1. Torchaudio: https://pytorch.org/audio/0.7.0/index.html
2. Torchtext: https://pytorch.org/text/stable/index.html
3. Torchvision: https://pytorch.org/vision/stable/index.html
4. TorchElastic: https://pytorch.org/elastic/0.2.2/index.html
5. TorchServe: https://pytorch.org/serve/
6. PyTorch for XLA devices: https://pytorch.org/xla/release/1.7/index.html

1. PyTorch Lightning: https://www.pytorchlightning.ai/

References you should use!

1. Official Website: https://pytorch.org
2. PyTorch Intro Tutorial: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
3. Fast-PyTorch: https://github.com/omerbsezer/Fast-Pytorch
4. Official Documentation: https://pytorch.org/docs/stable/index.html
5. https://www.analyticsvidhya.com/blog/2019/09/introduction-to-pytorch-from-scratch/