Introduction to PyTorch

GPU 2017

Anuvabh Dutt
anuvabh.dutt@univ-grenoble-alpes.fr

What is PyTorch?

- `Ndarray` library with GPU support
- Automatic differentiation engine
  - Gradient based optimisation package
- Neural network primitives (cuDNN integration)
- Utilities: Data loading, model zoo, distributed computation, etc.

Tensor library
`torch.Tensor`

- `torch.Tensor` is similar to ndarray in NumPy
- `np.ndarray` <-> `torch.Tensor` can be replaced in quite a few places
- A lot of operations defined for Tensors http://pytorch.org/docs/master/tensors.html
- Acceleration through CUDA

```
[In [26]: %timeit a + b
100 loops, best of 3: 2.64 ms per loop

[In [27]: %timeit a_thc + b_thc
10000 loops, best of 3: 125 us per loop
```

Automatic Differentiation through `autograd`

- Compute gradients for Tensor operations
- `X = Variable(torch.randn(10))`
- `W = Variable(torch.randn(10))`
- `B = Variable(torch.randn(1))`
- `T = Variable(torch.randn(1))`
- `Y = W.X + B`
- `L = Y - T`
- Gradient: dL / dW

Automatic Differentiation through `autograd`

- Wrap `Tensor` objects in `Variable`
- `autograd` tracks the sequence of operations applied to `Variables`
- Call `.backward()` to compute gradients
  - It is called `.backward()` because of the back propagation algorithm
- `Variable` has two important attributes:
  - `grad_fn: A Function` which created the `Variable`
  - `grad: Variable` holding the gradient

Automatic Differentiation through `autograd`

- `autograd` can be extended to include operations which are not defined
  - Look at implementations of existing functions: https://github.com/pytorch/pytorch/tree/master/torch/autograd/functions
  - Check correctness against numerical gradients: https://github.com/pytorch/pytorch/blob/master/torch/autograd/gradcheck.py

- Documentation: http://pytorch.org/docs/master/autograd.html

- The API will change soon. Wrapping in `Variable` will not be required.

Neural Networks
`torch.nn`

- Neural network primitives present in `torch.nn` package

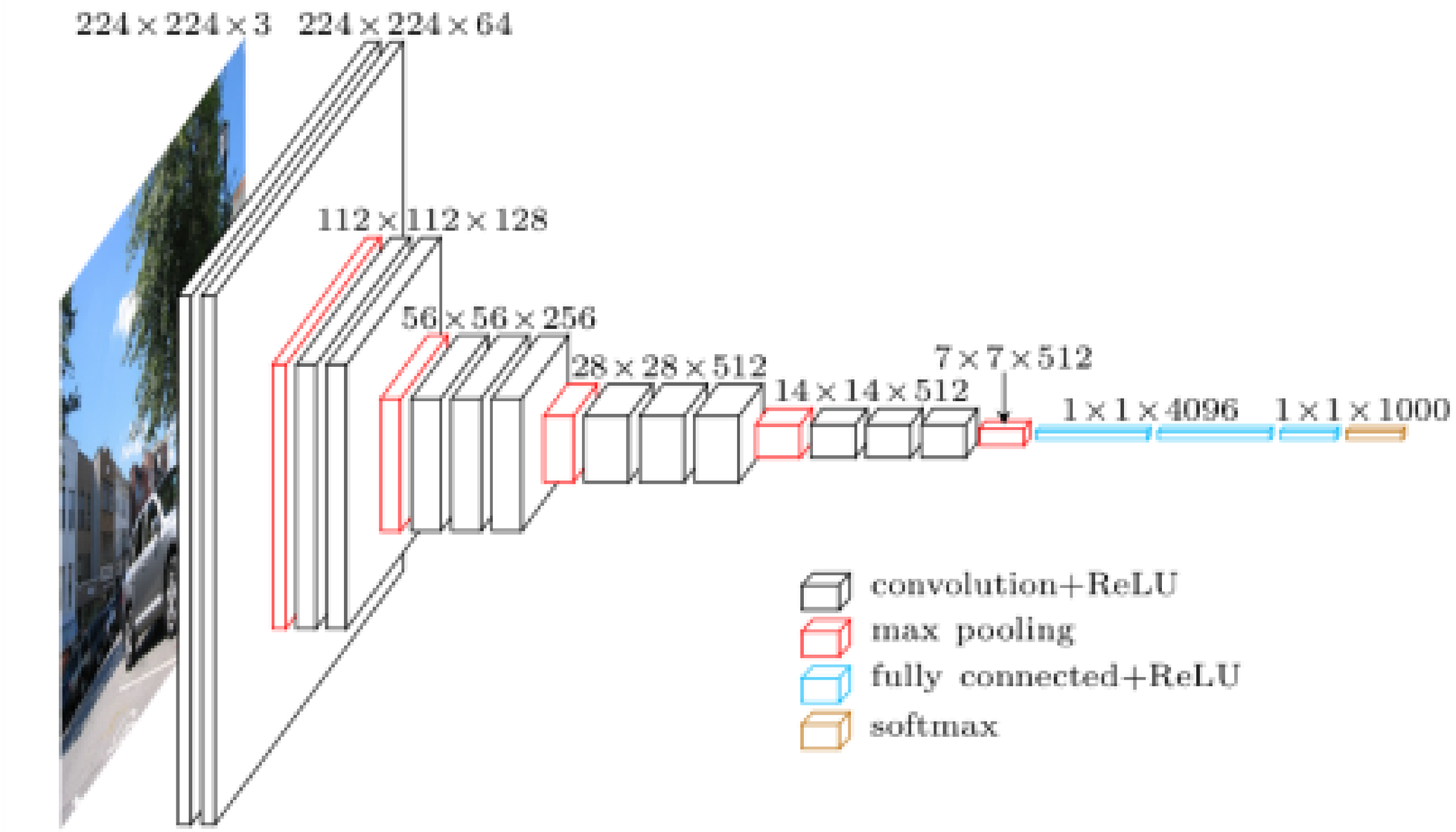
- `nn.Module` is used to encapsulate trainable parameters and define their operations

- `nn.Parameter` is a kind of `Variable` which is registered for *training*.

Neural Networks
`torch.nn`

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```


224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax

Recap

- `torch.Tensor` - A *multi-dimensional array*.
- `autograd.Variable` - *Wraps a Tensor and records the history of operations* applied to it. Has the same API as a `Tensor`, with some additions like `backward()`. Also *holds the gradient* w.r.t. the tensor.
- `nn.Module` - Neural network module. *Convenient way of encapsulating parameters*, with helpers for moving them to GPU, exporting, loading, etc.
- `nn.Parameter` - A kind of Variable, that is *automatically registered as a parameter when assigned as an attribute to a* `Module`.
- `autograd.Function` - Implements *forward and backward definitions of an autograd operation*. Every `Variable` operation, creates at least a single `Function` node, that connects to functions that created a `Variable` and *encodes its history*.

Next Steps

- Advanced examples: https://github.com/pytorch/examples
- Explore models: https://github.com/pytorch/vision/tree/master/torchvision/models
- Introductory DL course with coding assignments: http://cs231n.github.io
- Theory of DL: https://stats385.github.io

PyTorch Alternatives

- TensorFlow Eager
- CuPy Chainer (~99% NumPy replacement)
- MxNet