PyTorch

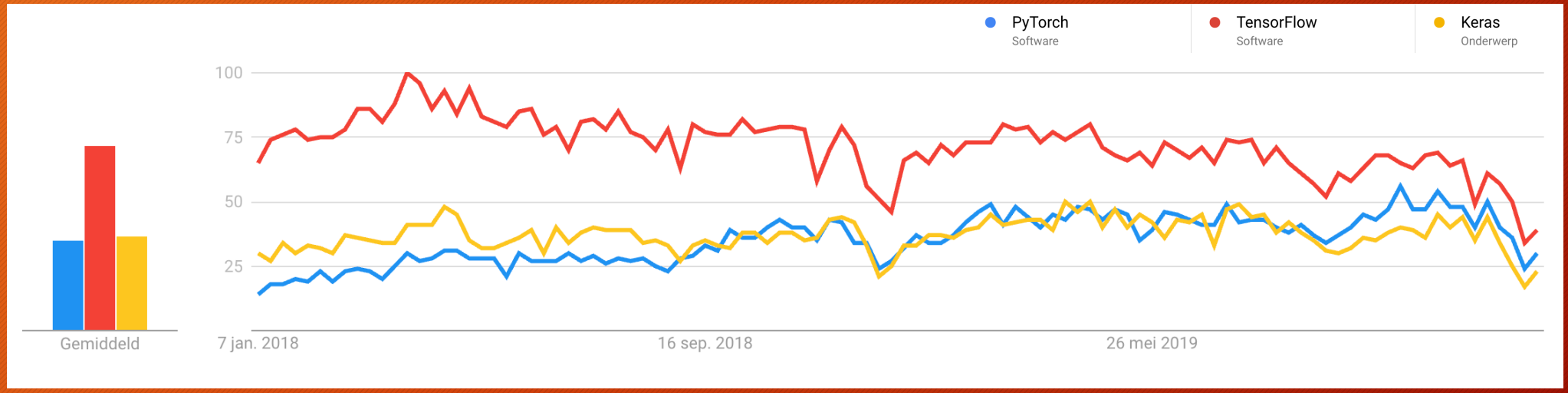
PyTorch

An introduction

PyTorch

- Open Source (Developed by Facebook)
- Not a Deep Learning library
  - Instead: Machine Learning framework that provides Automatic Differentiation
  - Can be used as GPU implementation of NumPy
- Extended/Encapsulated by many other libraries
  - KeOps
  - PyTorch Geometric
- Has interface levels of high and low abstraction
- Pythonic Framework

PyTorch vs. TensorFlow vs. Keras



PyTorch Execution

- Makes use of Dynamic Graphs (no compilation needed)
- Use torch.cuda for GPU semantics
- GPU operations are Asynchronous
  - Each device runs operations from a Queue/Stream
- Torch.autograd
  - data, grad, grad_fn
  - .backward(): accumulates the gradients

https://pytorch.org/docs/stable/notes/cuda.html

PyTorch Execution

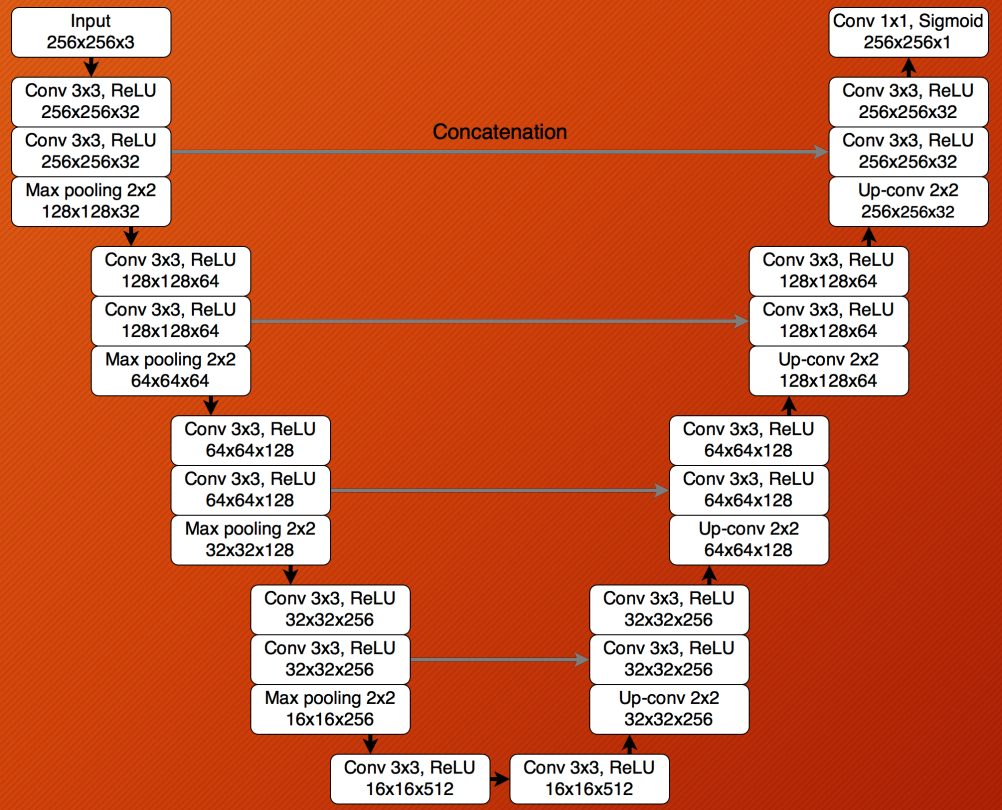
```
>>> x = torch.ones(2, 2, requires_grad = True)
>>> y = x + 2
>>> z = x + y
>>> out = z.mean()
>>> out
tensor(4., grad_fn=<MeanBackward0>)
>>> out.backward()
>>> x.grad
tensor([[0.5000, 0.5000],
        [0.5000, 0.5000]])
>>>
```

```python
class MyReLU(torch.autograd.Function):
    """
    We can implement our own custom autograd Functions by subclassing
    torch.autograd.Function and implementing the forward and backward passes
    which operate on Tensors.
    """

    @staticmethod
    def forward(ctx, input):
        """
        In the forward pass we receive a Tensor containing the input and return
        a Tensor containing the output. ctx is a context object that can be
used
        to stash information for backward computation. You can cache arbitrary
        objects for use in the backward pass using the ctx.save_for_backward
method.
        """
        ctx.save_for_backward(input)
        return input.clamp(min=0)

    @staticmethod
    def backward(ctx, grad_output):
        """
        In the backward pass we receive a Tensor containing the gradient of the
loss
        with respect to the output, and we need to compute the gradient of the
loss
        with respect to the input.
        """
        input, = ctx.saved_tensors
        grad_input = grad_output.clone()
        grad_input[input < 0] = 0
        return grad_input
```

Code Example



Code Example – Data Science Pipeline

Code Example – Model Development

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Code Example – Solving a Linear System

- KeOps

$$a^\star = \operatorname{argmin} \|(\alpha \operatorname{Id} + K_{xx})a - b\|_2^2, \quad \text{i.e.} \quad a^\star = (\alpha \operatorname{Id} + K_{xx})^{-1} b$$

```
x = torch.randn(M, D, requires_grad=True).type(tensor)  # Random point cloud
x_i = LazyTensor(x[:, None, :])  # (M, 1, D) LazyTensor
x_j = LazyTensor(x[None, :, :])  # (1, M, D) LazyTensor

K_xx = (- ((x_i - x_j) ** 2).sum(-1)).exp()  # Symbolic (M, M) Gaussian kernel matrix

alpha = .1  # "Ridge" regularization parameter
b_i = torch.randn(M, 4).type(tensor)  # Target signal, supported by the x_i's
a_i = K_xx.solve(b_i, alpha=alpha)  # Source signal, supported by the x_i's

print("a_i is now a {} of shape {}.".format(type(a_i), a_i.shape))
```

https://www.kernel-operations.io/keops/_auto_tutorials/a_LazyTensors/plot_lazytensors_b.html#sphx-glr-auto-tutorials-a-lazytensors-plot-lazytensors-b-py

Conclusion

- PyTorch is an intuitive machine learning Library that offers both a high and low level interface
- High performance intended to work with Large Datasets
- Very customizable

Q & A