Comparing TensorFlow 2.0 with PyTorch and PyTorch JIT

Tim Lazarus

29 November, 2019

Comparing TensorFlow 2.0 with PyTorch and PyTorch JIT

Tim Lazarus

29 November, 2019

Static Computation Graphs

In a *static execution model*, we define the graph structure at compile time.

The graph then gets built and compile-time optimisations are applied.

The same graph can be used multiple times.

TensorFlow is a prime example of a system with static computation graphs.

Dynamic Computation Graphs

In *dynamic execution models*, the graph is defined as it is run.

This means the structure of the graph can be dependent on the input.

This is exceptionally useful in the NLP domain, where loops in graphs are common.

PyTorch is a prime example of a system with dynamic computation graphs.

Example

A graph is created on the fly

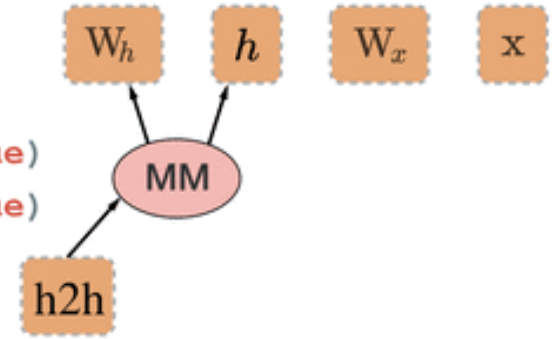

```
W_h = torch.randn(20, 20, requires_grad=True)
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```

Example

A graph is created on the fly

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

h2h = torch.mm(W_h, prev_h.t())
```

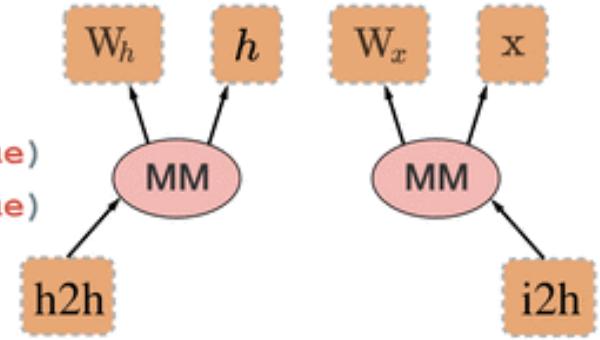


Example

A graph is created on the fly

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

h2h = torch.mm(W_h, prev_h.t())
i2h = torch.mm(W_x, x.t())
```

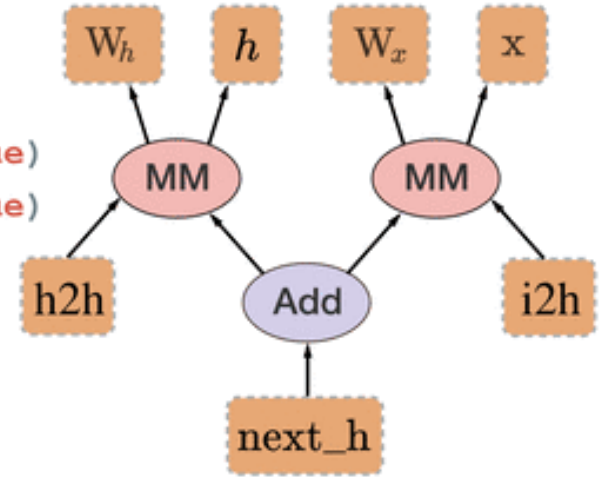


Example

A graph is created on the fly

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

h2h = torch.mm(W_h, prev_h.t())
i2h = torch.mm(W_x, x.t())
next_h = h2h + i2h
```

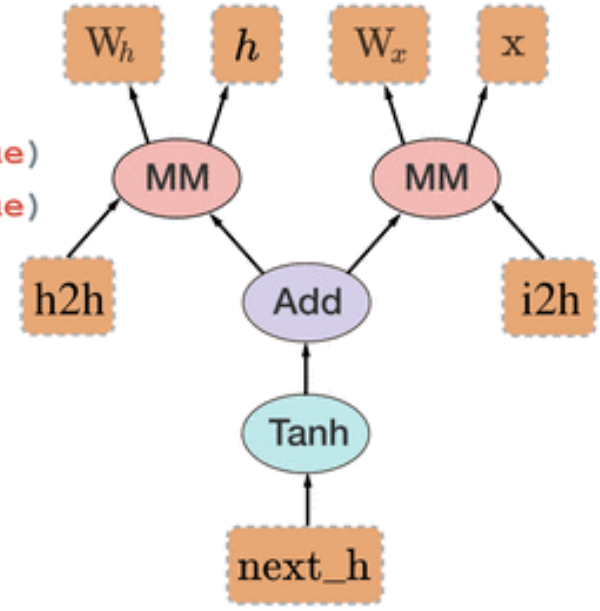


Example

A graph is created on the fly

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

h2h = torch.mm(W_h, prev_h.t())
i2h = torch.mm(W_x, x.t())
next_h = h2h + i2h
next_h = next_h.tanh()
```



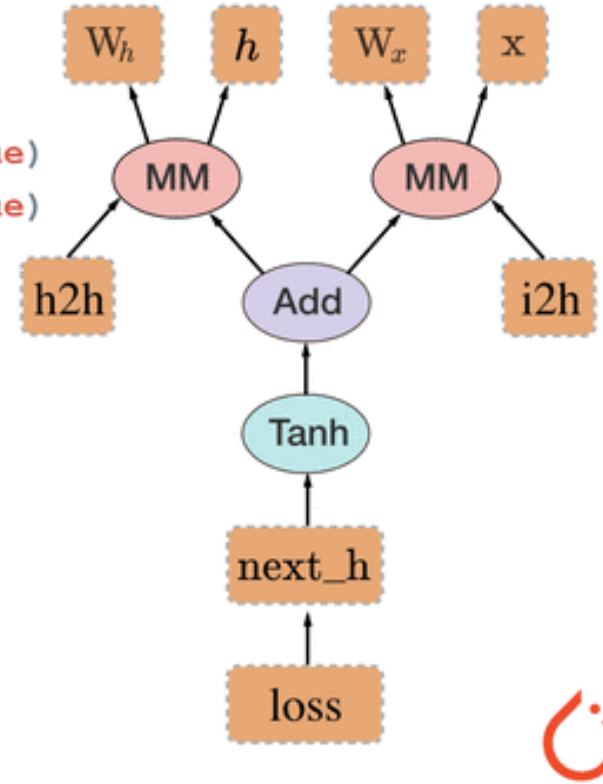
Example

A graph is created on the fly

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

h2h = torch.mm(W_h, prev_h.t())
i2h = torch.mm(W_x, x.t())
next_h = h2h + i2h
next_h = next_h.tanh()

loss = next_h.sum()
```



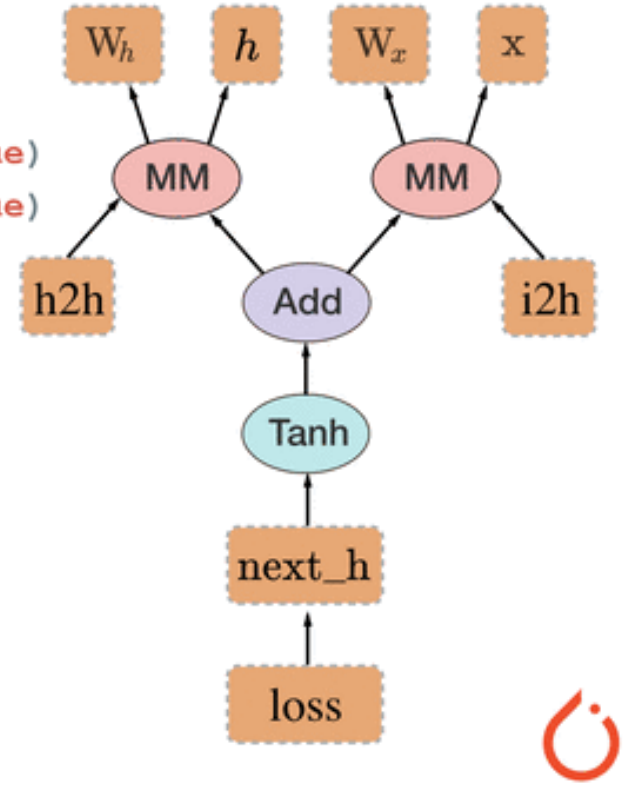
Example

Back-propagation
uses the dynamically created graph

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

h2h = torch.mm(W_h, prev_h.t())
i2h = torch.mm(W_x, x.t())
next_h = h2h + i2h
next_h = next_h.tanh()

loss = next_h.sum()
loss.backward()  # compute gradients!
```



PyTorch JIT

PyTorch actually has a *Just in Time Compiler*

At a basic level the JIT uses a subset of Python called *Torch Script* to define graphs so certain optimisations are possible and for increased portability.

For simple networks, this can be easily compiled using `trace`, but for more complex graphs with control flow the user must directly write in Torch Script.

The Project

TensorFlow 2.0, just introduced eager execution to compete with PyTorch.

I wish to compare the performance between TensorFlow 2.0, PyTorch and PyTorch with its JIT.

I would imagine that the JIT would outperform PyTorch, but I am interested to see the difference in the other systems.

Questions?