JIT scripting & Autocast

Overview

Autocast (aka Automatic Mixed Precision) is an optimization which helps taking advantage of the storage and performance benefits of narrow types (float16) while preserving the additional range and numerical precision of float32.

The JIT support for autocast is subject to different constraints compared to the eager mode implementation (mostly related to the fact that TorchScript is statically typed) and this document attempts to list the known limitations.

Usage

Explicit `with autocast()` scopes are supported inside scripted functions and modules (subject to the limitations described below):

```
import torch
from torch.cuda.amp import autocast

@torch.jit.script
def func(a, b):
    with autocast():
        return torch.mm(a, b)

a_float32 = torch.rand((8, 8), dtype=torch.float32, device="cuda")
b_float32 = torch.rand((8, 8), dtype=torch.float32, device="cuda")
result = func(a_float32, b_float32)
print(result.dtype) # expecting torch.float16
```

Known limitations

This section documents the current set of known limitations. Ideally this list will shrink as we advance with the design and implementation, although some of the limitations are related to fundamental TorchScript aspects that are not easy to change.

> One important goal is to avoid surprises (ex. autocast annotations silently ignored) and to report sensible diagnostics when something deviates from eager mode behavior.
>
> Please report any issues not covered here.

Diagnostics The current Autocast/JIT diagnostics should be improved: - Some errors are not specific enough or not actionable - Not all the errors point to the Python source location

Autocast decorators Using `@autocast` is not currently supported in script mode (a diagnostic will be emitted)

```
@autocast(enabled=True)
def helper(x):
    ...

@torch.jit.script
def foo(x):
    return helper(x) # not supported
```

Another example

```
@torch.jit.script
@autocast() # not supported
def foo(a, b, c, d):
    ...
```

Autocast argument must be a compile-time constant

```
@torch.jit.script
def fn(a, b, use_amp: bool):
    # runtime values for autocast enable argument are not supported
    with autocast(enabled=use_amp):
        return torch.mm(a, b)
```

Uncommon autocast usage patterns may not be supported

```
@torch.jit.script
def fn(a, b, c, d):
    with autocast(enabled=True) as autocast_instance: # not supported
        ...
```

```
        with autocast_instance:
            ...
```

Limited support for promote autocast policy For some operations, autocast needs to promote to the widest argument type. When the concrete types are not available, the current implementation will conservatively inject a promotion even when it may not be needed.

Missing autocast policies Also related to the lack of concrete dtype availability, a few specialized autocast policies are not yet supported with JIT scripting:
- CastPolicy::fp32_append_dtype

Mixing tracing and scripting autocast (script calling traced) Calling a traced function from a scripted one mostly works, except for the case where the traced part uses `autocast(False)`. After tracing, the `autocast` is stripped from the TorchScript IR so it's effectively ignored:

> This is one known limitation where we don't have a way to emit a diagnostic!

```
def helper(a, b):
    with autocast(enabled=False):
        return torch.mm(a, b) * 2.0

traced = torch.jit.trace(helper, (x, y))

@torch.jit.script
def fn(a, b):
    with autocast(enabled=True):
        return traced(a, b)
```

Mixing tracing and scripting autocast (traced calling script) Calling a scripted function from a trace is similar to calling the scripted function from eager mode:

```
@torch.jit.script
def fn(a, b):
    return torch.mm(a, b)

def traced(a, b):
    with autocast(enabled=True):
        return fn(a, b)

# running TorchScript with Autocast enabled is not supported
torch.jit.trace(traced, (x, y))
```

References

- torch.cuda.amp Package
- Automatic Mixed Precision - Tutorial
- Automatic Mixed Precision - Examples