torch 0.2.0 - Initial JIT support and many bug fixes

TORCH PACKAGES/RELEASES

The torch 0.2.0 release includes many bug fixes and some nice new features like initial JIT support, multi-worker dataloaders, new optimizers and a new print method for nn_modules.

AUTHOR
Daniel Falbel

AFFILIATION
RStudio

PUBLISHED
Dec. 14, 2020

CITATION
Falbel, 2020

Contents

We are happy to announce that the version 0.2.0 of `torch` just landed on CRAN.

This release includes many bug fixes and some nice new features that we will present in this blog post. You can see the full changelog in the NEWS.md file.

The features that we will discuss in detail are:

- Initial support for JIT tracing
- Multi-worker dataloaders
- Print methods for `nn_modules`

Multi-worker dataloaders

`dataloaders` now respond to the `num_workers` argument and will run the pre-processing in parallel workers.

For example, say we have the following dummy dataset that does a long computation:

```
library(torch)
dat <- dataset(
  "mydataset",
  initialize = function(time, len = 10) {
    self$time <- time
    self$len <- len
  },
  .getitem = function(i) {
    Sys.sleep(self$time)
    torch_randn(1)
  },
  .length = function() {
    self$len
  }
)
ds <- dat(1)
system.time(ds[1])
```

```
   user  system elapsed
  0.029   0.005   1.027
```

We will now create two dataloaders, one that executes sequentially and another executing in parallel.

```
seq_dl <- dataloader(ds, batch_size = 5)
par_dl <- dataloader(ds, batch_size = 5, num_workers = 2)
```

We can now compare the time it takes to process two batches sequentially to the time it takes in parallel:

```
seq_it <- dataloader_make_iter(seq_dl)
par_it <- dataloader_make_iter(par_dl)

two_batches <- function(it) {
  dataloader_next(it)
  dataloader_next(it)
  "ok"
}

system.time(two_batches(seq_it))
system.time(two_batches(par_it))
```

```
   user  system elapsed
  0.098   0.032  10.086
   user  system elapsed
  0.065   0.008   5.134
```

Note that it is batches that are obtained in parallel, not individual observations. Like that, we will be able to support datasets with variable batch sizes in the future.

Using multiple workers is **not** necessarily faster than serial execution because there's a considerable overhead when passing tensors from a worker to the main session as well as when initializing the workers.

This feature is enabled by the powerful `callr` package and works in all operating systems supported by `torch`. `callr` let's us create persistent R sessions, and thus, we only pay once the overhead of transferring potentially large dataset objects to workers.

In the process of implementing this feature we have made dataloaders behave like `coro` iterators. This means that you can now use `coro`'s syntax for looping through the dataloaders:

```
coro::loop(for(batch in par_dl) {
  print(batch$shape)
})
```

```
[1] 5 1
[1] 5 1
```

This is the first `torch` release including the multi-worker dataloaders feature, and you might run into edge cases when using it. Do let us know if you find any problems.

Initial JIT support

Programs that make use of the `torch` package are inevitably R programs and thus, they always need an R installation in order to execute.

As of version 0.2.0, `torch` allows users to JIT *trace* `torch` R functions into TorchScript. JIT (Just in time) tracing will invoke an R function with example inputs, record all operations that occured when the function was run and return a `script_function` object containing the TorchScript representation.

The nice thing about this is that TorchScript programs are easily serializable, optimizable, and they can be loaded by another program written in PyTorch or LibTorch without requiring any R dependency.

Suppose you have the following R function that takes a tensor, and does a matrix multiplication with a fixed weight matrix and then adds a bias term:

```
w <- torch_randn(10, 1)
```

```
b <- torch_randn(1)
fn <- function(x) {
  a <- torch_mm(x, w)
  a + b
}
```

This function can be JIT-traced into TorchScript with `jit_trace` by passing the function and example inputs:

```
x <- torch_ones(2, 10)
tr_fn <- jit_trace(fn, x)
tr_fn(x)
```

```
torch_tensor
-0.6880
-0.6880
[ CPUFloatType{2,1} ]
```

Now all `torch` operations that happened when computing the result of this function were traced and transformed into a graph:

```
tr_fn$graph
```

```
graph(%0 : Float(2:10, 10:1, requires_grad=0, device=cpu)):
  %1 : Float(10:1, 1:1, requires_grad=0, device=cpu) = prim::Constant[value=-0.3532  0.6490
  %2 : Float(2:1, 1:1, requires_grad=0, device=cpu) = aten::mm(%0, %1)
  %3 : Float(1:1, requires_grad=0, device=cpu) = prim::Constant[value={-0.558343}]()
  %4 : int = prim::Constant[value=1]()
  %5 : Float(2:1, 1:1, requires_grad=0, device=cpu) = aten::add(%2, %3, %4)
  return (%5)
```

The traced function can be serialized with `jit_save`:

```
jit_save(tr_fn, "linear.pt")
```

It can be reloaded in R with `jit_load`, but it can also be reloaded in Python with `torch.jit.load`:

```
import torch
fn = torch.jit.load("linear.pt")
fn(torch.ones(2, 10))
```

```
tensor([[-0.6880],
        [-0.6880]])
```

How cool is that?!

This is just the initial support for JIT in R. We will continue developing this. Specifically, in the next version of `torch` we plan to support tracing `nn_modules` directly. Currently, you need to detach all parameters before tracing them; see an example here. This will allow you also to take benefit of TorchScript to make your models run faster!

Also note that tracing has some limitations, especially when your code has loops or control flow statements that depend on tensor data. See `?jit_trace` to learn more.

New print method for `nn_modules`

In this release we have also improved the `nn_module` printing methods in order to make it easier to understand what's inside.

For example, if you create an instance of an `nn_linear` module you will see:

```
nn_linear(10, 1)
```

```
An `nn_module` containing 11 parameters.

── Parameters ──────────────────────────────────────────────────────────────────
● weight: Float [1:1, 1:10]
● bias: Float [1:1]
```

You immediately see the total number of parameters in the module as well as their names and shapes.

This also works for custom modules (possibly including sub-modules). For example:

```
my_module <- nn_module(
  initialize = function() {
    self$linear <- nn_linear(10, 1)
    self$param <- nn_parameter(torch_randn(5,1))
    self$buff <- nn_buffer(torch_randn(5))
  }
)
my_module()
```

```
An `nn_module` containing 16 parameters.

── Modules ─────────────────────────────────────────────────────────────────────
● linear: <nn_linear> #11 parameters

── Parameters ──────────────────────────────────────────────────────────────────
● param: Float [1:5, 1:1]
```

```
── Buffers ─────────────────────────────────────────────────────────────────────
● buff: Float [1:5]
```

We hope this makes it easier to understand `nn_module` objects. We have also improved autocomplete support for `nn_modules` and we will now show all sub-modules, parameters and buffers while you type.

torchaudio

`torchaudio` is an extension for `torch` developed by Athos Damiani (`@athospd`), providing audio loading, transformations, common architectures for signal processing, pre-trained weights and access to commonly used datasets. An almost literal translation from PyTorch's Torchaudio library to R.

`torchaudio` is not yet on CRAN, but you can already try the development version available here.

You can also visit the `pkgdown` website for examples and reference documentation.

Other features and bug fixes

Thanks to community contributions we have found and fixed many bugs in `torch`. We have also added new features including:

- `element_size` and `bool` Tensor methods by `@dirkschumacher`
- checking the MD5 hashes of downloaded LibTorch binaries by `@dirkschumacher`
- initial development for the Distributions module by `@krzjoa`
- the `nn_batch_norm3d` module implemented by `@mattwarkentin`
- a Dockerfile with GPU support as well as an installation guide by `@y-vectorfield`

You can see the full list of changes in the NEWS.md file.

Thanks very much for reading this blog post, and feel free to reach out on GitHub for help or discussions!

The photo used in this post preview is by Oleg Illarionov on Unsplash

Reuse

Citation

For attribution, please cite this work as

```
Falbel (2020, Dec. 15). RStudio AI Blog: torch 0.2.0 - Initial JIT support and many bug fixes.
Retrieved from https://blogs.rstudio.com/tensorflow/posts/2020-12-15-torch-0.2.0-released/
```

BibTeX citation

```
@misc{torchzero2,
  author = {Falbel, Daniel},
  title = {RStudio AI Blog: torch 0.2.0 - Initial JIT support and many bug fixes},
  url = {https://blogs.rstudio.com/tensorflow/posts/2020-12-15-torch-0.2.0-released/},
  year = {2020}
}
```