

---

## tch-rs

Rust bindings for the C++ api of PyTorch. The goal of the `tch` crate is to provide some thin wrappers around the C++ PyTorch api (a.k.a. libtorch). It aims at staying as close as possible to the original C++ api. More idiomatic rust bindings could then be developed on top of this. The documentation can be found on docs.rs.



changelog

The code generation part for the C api on top of libtorch comes from ocaml-torch.

### Getting Started

This crate requires the C++ PyTorch library (libtorch) in version v2.3.0 to be available on your system. You can either:

- Use the system-wide libtorch installation (default).
- Install libtorch manually and let the build script know about it via the `LIBTORCH` environment variable.
- Use a Python PyTorch install, to do this set `LIBTORCH_USE_PYTORCH=1`.
- When a system-wide libtorch can't be found and `LIBTORCH` is not set, the build script can download a pre-built binary version of libtorch by using the `download-libtorch` feature. By default a CPU version is used. The `TORCH_CUDA_VERSION` environment variable can be set to `cu117` in order to get a pre-built binary using CUDA 11.7.

### System-wide Libtorch

On linux platforms, the build script will look for a system-wide libtorch library in `/usr/lib/libtorch.so`.

### Python PyTorch Install

If the `LIBTORCH_USE_PYTORCH` environment variable is set, the active python interpreter is called to retrieve information about the torch python package. This version is then linked against.

---

## Libtorch Manual Install

- Get `libtorch` from the PyTorch website download section and extract the content of the zip file.
- For Linux and macOS users, add the following to your `.bashrc` or equivalent, where `/path/to/libtorch` is the path to the directory that was created when unzipping the file.

```
1 export LIBTORCH=/path/to/libtorch
```

The header files location can also be specified separately from the shared library via the following:

```
1 # LIBTORCH_INCLUDE must contain `include` directory.
2 export LIBTORCH_INCLUDE=/path/to/libtorch/
3 # LIBTORCH_LIB must contain `lib` directory.
4 export LIBTORCH_LIB=/path/to/libtorch/
```

- For Windows users, assuming that `X:\path\to\libtorch` is the unzipped libtorch directory.
  - Navigate to Control Panel -> View advanced system settings -> Environment variables.
  - Create the `LIBTORCH` variable and set it to `X:\path\to\libtorch`.
  - Append `X:\path\to\libtorch\lib` to the `Path` variable.

If you prefer to temporarily set environment variables, in PowerShell you can run

```
1 $Env:LIBTORCH = "X:\path\to\libtorch"
2 $Env:Path += ";X:\path\to\libtorch\lib"
```

- You should now be able to run some examples, e.g. `cargo run --example basics`.

## Windows Specific Notes

As per the pytorch docs the Windows debug and release builds are not ABI-compatible. This could lead to some segfaults if the incorrect version of libtorch is used.

It is recommended to use the MSVC Rust toolchain (e.g. by installing `stable-x86_64-pc-windows-msvc` via rustup) rather than a MinGW based one as PyTorch has compatibilities issues with MinGW.

## Static Linking

When setting environment variable `LIBTORCH_STATIC=1`, `libtorch` is statically linked rather than using the dynamic libraries. The pre-compiled artifacts don't seem to include `libtorch.a`

---

by default so this would have to be compiled manually, e.g. via the following:

```
1 git clone -b v2.3.0 --recurse-submodule https://github.com/pytorch/
  pytorch.git pytorch-static --depth 1
2 cd pytorch-static
3 USE_CUDA=OFF BUILD_SHARED_LIBS=OFF python setup.py build
4 # export LIBTORCH to point at the build directory in pytorch-static.
```

## Examples

### Basic Tensor Operations

This crate provides a tensor type which wraps PyTorch tensors. Here is a minimal example of how to perform some tensor operations.

```
1 use tch::Tensor;
2
3 fn main() {
4     let t = Tensor::from_slice(&[3, 1, 4, 1, 5]);
5     let t = t * 2;
6     t.print();
7 }
```

### Training a Model via Gradient Descent

PyTorch provides automatic differentiation for most tensor operations it supports. This is commonly used to train models using gradient descent. The optimization is performed over variables which are created via a `nn::VarStore` by defining their shapes and initializations.

In the example below `my_module` uses two variables `x1` and `x2` which initial values are 0. The forward pass applied to tensor `xs` returns `xs * x1 + exp(xs) * x2`.

Once the model has been generated, a `nn::Sgd` optimizer is created. Then on each step of the training loop:

- The forward pass is applied to a mini-batch of data.
- A loss is computed as the mean square error between the model output and the mini-batch ground truth.
- Finally an optimization step is performed: gradients are computed and variables from the `VarStore` are modified accordingly.

```
1 use tch::nn::{Module, OptimizerConfig};
2 use tch::{kind, nn, Device, Tensor};
```

```

3
4 fn my_module(p: nn::Path, dim: i64) -> impl nn::Module {
5     let x1 = p.zeros("x1", &[dim]);
6     let x2 = p.zeros("x2", &[dim]);
7     nn::func(move |xs| xs * &x1 + xs.exp() * &x2)
8 }
9
10 fn gradient_descent() {
11     let vs = nn::VarStore::new(Device::Cpu);
12     let my_module = my_module(vs.root(), 7);
13     let mut opt = nn::Sgd::default().build(&vs, 1e-2).unwrap();
14     for _idx in 1..50 {
15         // Dummy mini-batches made of zeros.
16         let xs = Tensor::zeros(&[7], kind::FLOAT_CPU);
17         let ys = Tensor::zeros(&[7], kind::FLOAT_CPU);
18         let loss = (my_module.forward(&xs) - ys).pow_tensor_scalar(2).
19                     sum(kind::Kind::Float);
20         opt.backward_step(&loss);
21     }
22 }

```

## Writing a Simple Neural Network

The `nn` api can be used to create neural network architectures, e.g. the following code defines a simple model with one hidden layer and trains it on the MNIST dataset using the Adam optimizer.

```

1 use anyhow::Result;
2 use tch::{nn, nn::Module, nn::OptimizerConfig, Device};
3
4 const IMAGE_DIM: i64 = 784;
5 const HIDDEN_NODES: i64 = 128;
6 const LABELS: i64 = 10;
7
8 fn net(vs: &nn::Path) -> impl Module {
9     nn::seq()
10         .add(nn::linear(
11             vs / "layer1",
12             IMAGE_DIM,
13             HIDDEN_NODES,
14             Default::default(),
15         ))
16         .add_fn(|xs| xs.relu())
17         .add(nn::linear(vs, HIDDEN_NODES, LABELS, Default::default()))
18 }
19
20 pub fn run() -> Result<> {
21     let m = tch::vision::mnist::load_dir("data")?;
22     let vs = nn::VarStore::new(Device::Cpu);
23     let net = net(&vs.root());

```

---

```

24     let mut opt = nn::Adam::default().build(&vs, 1e-3)?;
25     for epoch in 1..200 {
26         let loss = net
27             .forward(&m.train_images)
28             .cross_entropy_for_logits(&m.train_labels);
29         opt.backward_step(&loss);
30         let test_accuracy = net
31             .forward(&m.test_images)
32             .accuracy_for_logits(&m.test_labels);
33         println!(
34             "epoch: {:4} train loss: {:.8.5} test acc: {:.5.2}%",
35             epoch,
36             f64::from(&loss),
37             100. * f64::from(&test_accuracy),
38         );
39     }
40     Ok(())
41 }

```

More details on the training loop can be found in the detailed tutorial.

## Using some Pre-Trained Model

The pretrained-models example illustrates how to use some pre-trained computer vision model on an image. The weights - which have been extracted from the PyTorch implementation - can be downloaded [here](#) resnet18.ot and [here](#) resnet34.ot.

The example can then be run via the following command:

```
1 cargo run --example pretrained-models -- resnet18.ot tiger.jpg
```

This should print the top 5 imagenet categories for the image. The code for this example is pretty simple.

```

1     // First the image is loaded and resized to 224x224.
2     let image = imagenet::load_image_and_resize(image_file)?;
3
4     // A variable store is created to hold the model parameters.
5     let vs = tch::nn::VarStore::new(tch::Device::Cpu);
6
7     // Then the model is built on this variable store, and the weights
8     // are loaded.
9     let resnet18 = tch::vision::resnet::resnet18(vs.root(), imagenet::
10         CLASS_COUNT);
11     vs.load(weight_file)?;
12
13     // Apply the forward pass of the model to get the logits and
14     // convert them

```

---

```

12 // to probabilities via a softmax.
13 let output = resnet18
14   .forward_t(&image.unsqueeze(0), /*train=*/ false)
15   .softmax(-1);
16
17 // Finally print the top 5 categories and their associated
18   probabilities.
19 for (probability, class) in imagenet::top(&output, 5).iter() {
20   println!("{:50} {:.5.2}%", class, 100.0 * probability)
21 }

```

## Importing Pre-Trained Weights from PyTorch Using SafeTensors

`safetensors` is a new simple format by HuggingFace for storing tensors. It does not rely on Python's `pickle` module, and therefore the tensors are not bound to the specific classes and the exact directory structure used when the model is saved. It is also zero-copy, which means that reading the file will require no more memory than the original file.

For more information on `safetensors`, please check out <https://github.com/huggingface/safetensors>

**Installing `safetensors`** You can install `safetensors` via the pip manager:

```
1 pip install safetensors
```

### Exporting weights in PyTorch

```

1 import torchvision
2 from safetensors import torch as stt
3
4 model = torchvision.models.resnet18(pretrained=True)
5 stt.save_file(model.state_dict(), 'resnet18.safetensors')

```

*Note: the filename of the export must be named with a `.safetensors` suffix for it to be properly decoded by `tch`.*

### Importing weights in `tch`

```

1 use anyhow::Result;
2 use tch::{
3   Device,
4   Kind,
5   nn::VarStore,
6   vision::{
7     imagenet,
8     resnet::resnet18,
9   }
10 };

```

---

```

11
12 fn main() -> Result<()> {
13     // Create the model and load the pre-trained weights
14     let mut vs = VarStore::new(Device::cuda_if_available());
15     let model = resnet18(&vs.root(), 1000);
16     vs.load("resnet18.safetensors"?);
17
18     // Load the image file and resize it to the usual imagenet
19     // dimension of 224x224.
20     let image = imagenet::load_image_and_resize224("dog.jpg")?
21         .to_device(vs.device());
22
23     // Apply the forward pass of the model to get the logits
24     let output = image
25         .unsqueeze(0)
26         .apply_t(&model, false)
27         .softmax(-1, Kind::Float);
28
29     // Print the top 5 categories for this image.
30     for (probability, class) in imagenet::top(&output, 5).iter() {
31         println!("{:50} {:.2}%", class, 100.0 * probability)
32     }
33     Ok(())
34 }

```

Further examples include: \* A simplified version of char-rnn illustrating character level language modeling using Recurrent Neural Networks. \* Neural style transfer uses a pre-trained VGG-16 model to compose an image in the style of another image (pre-trained weights: vgg16.ot). \* Some ResNet examples on CIFAR-10. \* A tutorial showing how to deploy/run some Python trained models using TorchScript JIT. \* Some Reinforcement Learning examples using the OpenAI Gym environment. This includes a policy gradient example as well as an A2C implementation that can run on Atari games. \* A Transfer Learning Tutorial shows how to finetune a pre-trained ResNet model on a very small dataset. \* A simplified version of GPT similar to minGPT. \* A Stable Diffusion implementation following the lines of huggingface's diffusers library.

External material: \* A tutorial showing how to use Torch to compute option prices and greeks. \* tchrs-opencv-webcam-inference uses [tch-rs](#) and [opencv](#) to run inference on a webcam feed for some Python trained model based on mobilenet v3.

## FAQ

### What are the best practices for Python to Rust model translations?

See some details in this thread.

---

### How to get this to work on a M1/M2 mac?

Check this issue.

### Compilation is slow, torch-sys seems to be rebuilt every time cargo gets run.

See this issue, this could be caused by rust-analyzer not knowing about the proper environment variables like `LIBTORCH` and `LD_LIBRARY_PATH`.

### Using Rust/tch code from Python.

It is possible to call Rust/tch code from Python via PyO3, tch-ext provides an example of such a Python extension.

### Error loading shared libraries.

If you get an error about not finding some shared libraries when running the generated binaries (e.g. `error while loading shared libraries: libtorch_cpu.so: cannot open shared object file: No such file or directory`). You can try adding the following to your `.bashrc` where `/path/to/libtorch` is the path to your libtorch install.

```
1 # For Linux
2 export LD_LIBRARY_PATH=/path/to/libtorch/lib:$LD_LIBRARY_PATH
3 # For macOS
4 export DYLD_LIBRARY_PATH=/path/to/libtorch/lib:$DYLD_LIBRARY_PATH
```

### License

`tch-rs` is distributed under the terms of both the MIT license and the Apache license (version 2.0), at your option.

See LICENSE-APACHE, LICENSE-MIT for more details.