**Rust AI**

[[Gautam Goswami](https://medium.com/@gautamgoswami?source=post_page-----81ed78dc5394--------------------------------)](https://medium.com/@gautamgoswami?source=post_page-----81ed78dc5394--------------------------------)

[Gautam Goswami](https://medium.com/@gautamgoswami?source=post_page-----81ed78dc5394--------------------------------)

·

Follow

4 min read

·

Nov 30

26

**TL;DR:** Drafting my learning of using Rust for AI.

**Setting up rust on Mac OS**

# Download rustup, add to the PATH and update it   
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh  
export PATH="$HOME/.cargo/bin:$PATH"  
rustup update  
rustc --version  
cargo --version

If everything goes fine, you have rust setup successfully.

**Setting up PyTorch with Rust**

Setup a new project

cargo new my\_pytorch\_rust

In order to use PyTorch in Rust, you need tch(short form of torch) in Cargo.toml (A config file to manage dependencies and their versions)

[package]  
name = "my\_torch\_project"  
version = "0.1.0"  
edition = "2021"  
  
# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html  
  
[dependencies]  
tch = "0.14" # Check https://crates.io/crates/tch for the latest version

tch is a rust binding for C++ API of official PyTorch.

You need to make a note of PyTorch version which is being supported by tch specific version. i.e. there is a 1:1 to mapping of tch and PyTorch.

The way you install it is by downloading the libtorch zip folder and updating the path accordingly.

export LIBTORCH='/Users/gautam/projects/rust-ai/libtorch'  
# LIBTORCH\_INCLUDE must contains `include` directory.  
export LIBTORCH\_INCLUDE='/Users/gautam/projects/rust-ai/libtorch'  
# LIBTORCH\_LIB must contains `lib` directory.  
export LIBTORCH\_LIB='/Users/gautam/projects/rust-ai/libtorch'  
# DYLD\_LIBRARY\_PATH is same as UNIX version of LD\_LIBRARY\_PATH  
export DYLD\_LIBRARY\_PATH=/Users/gautam/projects/rust-ai/libtorch/lib:$DYLD\_LIBRARY\_PATH

*If you notice you can have different version of PyTorch for different projects by adding their LIBTORCH on different path.*

Lets run a basic rust program of gradient descent and

Cargo.toml has tch v0.13 is corresponding to PyTorch v2.0.0. See [documentation](https://docs.rs/crate/tch/0.13.0) for details

[package]  
name = "my\_torch\_project"  
version = "0.1.0"  
edition = "2021"  
  
# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html  
[dependencies]  
anyhow = "1.0"  
tch = "0.13" # Check https://crates.io/crates/tch for the latest version,

Lets write main.rs for simple gradient descent. This code is available in [example](https://github.com/LaurentMazare/tch-rs#examples)

use anyhow::Result;  
use tch::nn::{Module, OptimizerConfig};  
use tch::{kind, nn, Device, Tensor};  
  
const IMAGE\_DIM: i64 = 784;  
const HIDDEN\_NODES: i64 = 128;  
const LABELS: i64 = 10;  
  
fn my\_module(p: nn::Path, dim: i64) -> impl nn::Module {  
 // Correct the arguments for randn  
 let x1 = p.randn("x1", &[dim], 0.0, 0.01); // mean = 0.0, stdev = 0.01  
 let x2 = p.randn("x2", &[dim], 0.0, 0.01); // mean = 0.0, stdev = 0.01  
 nn::func(move |xs| xs \* &x1 + xs.exp() \* &x2)  
}  
  
fn gradient\_descent() {  
 let vs = nn::VarStore::new(Device::Cpu);  
 let my\_module = my\_module(vs.root(), 7);  
 let mut opt = nn::Sgd::default().build(&vs, 1e-3).unwrap();  
   
 for idx in 1..50 {  
 // Use random values instead of zeros for xs and ys  
 let xs = Tensor::randn(&[7], kind::FLOAT\_CPU);  
 let ys = Tensor::randn(&[7], kind::FLOAT\_CPU);  
 let loss = (my\_module.forward(&xs) - ys).pow\_tensor\_scalar(2).sum(kind::Kind::Float);  
   
 // Print the current loss  
 println!("Step {}: Loss = {:?}", idx, loss);  
  
 // Perform a backward pass and optimizer step  
 opt.backward\_step(&loss);  
  
 // Optionally print the parameters  
 if idx % 5 == 0 { // Print every 10 steps  
 println!("Parameters at Step {}:", idx);  
 for (name, param) in vs.variables().iter() {  
 println!("{}: {:?}", name, param);  
 }  
 }  
 }  
}  
  
  
  
fn net(vs: &nn::Path) -> impl Module {  
 nn::seq()  
 .add(nn::linear(  
 vs / "layer1",  
 IMAGE\_DIM,  
 HIDDEN\_NODES,  
 Default::default(),  
 ))  
 .add\_fn(|xs| xs.relu())  
 .add(nn::linear(vs, HIDDEN\_NODES, LABELS, Default::default()))  
}  
  
// Run MNIST based example   
pub fn run() -> Result<()> {  
 let m = tch::vision::mnist::load\_dir("./data")?;  
 let vs = nn::VarStore::new(Device::Cpu);  
 let net = net(&vs.root());  
 let mut opt = nn::Adam::default().build(&vs, 1e-3)?;  
 for epoch in 1..1000 {  
 let loss = net  
 .forward(&m.train\_images)  
 .cross\_entropy\_for\_logits(&m.train\_labels);  
 opt.backward\_step(&loss);  
 let test\_accuracy = net  
 .forward(&m.test\_images)  
 .accuracy\_for\_logits(&m.test\_labels);  
  
  
 // Extract a scalar value from the tensor and convert it to f64  
 let loss\_value = loss.double\_value(&[]);  
 let test\_accuracy\_value = test\_accuracy.double\_value(&[]);  
 println!(  
 "epoch: {:4} train loss: {:8.5} test acc: {:5.2}%",  
 epoch,  
 loss\_value,  
 100. \* test\_accuracy\_value,  
 );  
 }  
 Ok(())  
}  
  
fn main() {  
 let t = Tensor::from\_slice(&[3, 1, 4, 1, 5]);  
 let t = t \* 2;  
 t.print();  
  
 gradient\_descent();  
  
 //Call the run function  
 if let Err(e) = run() {  
 eprintln!("Error: {:?}", e);  
 }  
}

Once you run cargo clean && cargo build && cargo run from main folder. you will see bunch of compilation

$ cargo run   
 Finished dev [unoptimized + debuginfo] target(s) in 0.29s  
 Running `target/debug/my\_torch\_project`  
 6  
 2  
 8  
 2  
 10  
[ CPUIntType{5} ]  
Step 1: Loss = [3.0695438385009766]  
Step 2: Loss = [14.553200721740723]  
Step 3: Loss = [5.786465167999268]  
Step 4: Loss = [3.288952350616455]  
Step 5: Loss = [9.384751319885254]  
Parameters at Step 5:  
x1: [-0.0040474142879247665, -0.01436722744256258, -0.01139893103390932, 0.023268481716513634, -0.008363340981304646, -0.005767303053289652, 0.020539185032248497]  
x2: [0.0056999316439032555, 0.009185446426272392, -0.02222108654677868, -0.004255229607224464, -0.018071794882416725, 0.004302480723708868, -0.0015405125450342894]  
Step 6: Loss = [19.099533081054688]  
  
Step 49: Loss = [7.96804666519165]  
epoch: 1 train loss: 2.35825 test acc: 14.09%  
epoch: 2 train loss: 2.21400 test acc: 32.79%  
epoch: 3 train loss: 2.08496 test acc: 48.62%  
epoch: 4 train loss: 1.96546 test acc: 58.54%

Lets make it more interesting. Lets use bert model to do sentiment analysis.

Create a new project sentiment\_analysis add following to Cargo.toml

[package]  
name = "sentiment\_analysis"  
version = "0.1.0"  
edition = "2021"  
  
# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html  
  
[dependencies]  
tch = "0.13" # Check https://crates.io/crates/tch for the latest version  
rust-bert = "0.21.0" # Check for the latest version

When you run cargo build first time it will download its dependencies and also pre-trained bert mode which we will use to do sentiment analysis. Again keep in mind you need to match libtorch version (equivalent to PyTorch) which is supported in rust-bert

main.rs

use rust\_bert::pipelines::sentiment::SentimentModel;  
  
fn main() -> Result<(), Box<dyn std::error::Error>> {  
 let sentiment\_model = SentimentModel::new(Default::default())?;  
 let input = ["I am happy today.", "The weather is terrible today."];  
 let output = sentiment\_model.predict(&input);  
  
 for sentiment in output {  
 println!("{:?}", sentiment);  
 }  
  
 Ok(())  
}

cargo run will result in following

Finished dev [unoptimized + debuginfo] target(s) in 0.67s  
 Running `target/debug/sentiment\_analysis`  
Sentiment { polarity: Positive, score: 0.9998859167098999 }  
Sentiment { polarity: Negative, score: 0.9986808896064758 }

*This was a day-1 learning of my introduction to rust.*