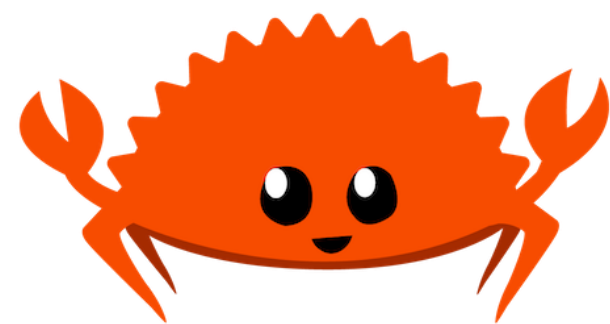
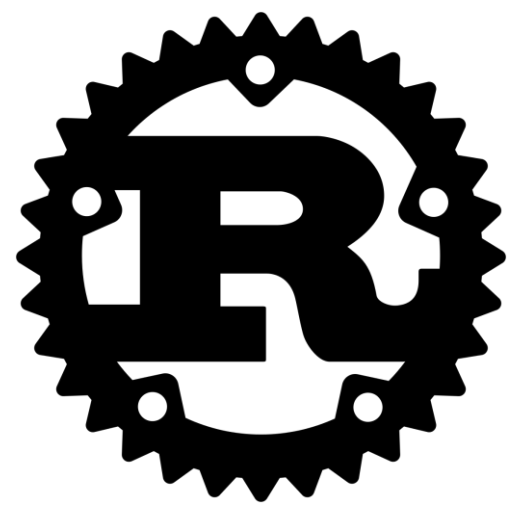




Knight Foundation School of computing & Information Sciences

Spring 2023 capstone 2 Senior Design Project

Deep Learning Framework in Rust

Students: Ngang Bah

Instructor/Faculty: Dr. Masoud Sadjadi, Florida International University

PROBLEM

The lack of a high-performing and efficient Rust-based library for machine learning is a significant challenge in the field. Rust is an excellent language for developing high-performance software due to its memory safety and thread-safety features, making it an ideal choice for implementing machine learning algorithms. However, existing libraries built on Rust are often incomplete or lack essential features, which creates a gap that this Deep learning framework aims to address.

CURRENT SYSTEM

Many of the popular machine learning frameworks like TensorFlow and Porch are primarily built in Python and C++. While these are great languages in terms of flexibility and expressiveness, Rust has a fast execution speed while maintaining the safety and reliability of memory management. This system was built from scratch, with Andrej Karpathy's micrograd as a foundational inspiration and optimized for Rust's capabilities.

REQUIREMENTS

- Capable of performing common machine learning tasks such as training and testing neural networks
- Support a variety of neural network architectures, including Linear, Pooling, and Convolutional layers.
- Provide seamless integration with other machine learning tools and libraries. Eg making sure APIs are similar to popular frameworks.

SCREENSHOTS

```
fn state_dict(&self) -> IndexMap<String, Vec<f64>> {
    let mut state_dict: IndexMap<String, Vec<f64>> = IndexMap::new();

    let trainable_layers: impl Iterator<Item = &Box<dyn Layer<D>>> = self.layers.iter().filter(|&v| &Box<dyn Layer<D>>| v.is_trainable);

    for layer: &Box<dyn Layer<D>> in trainable_layers {
        let (weight_key: String, bias_key: String) = (layer.name() + ".weight", layer.name() + ".bias");

        let layer_weights: Vec<f64> = layer.weights().iter().map(|&v| &Value v.value()).collect();
        state_dict.insert(weight_key, value: layer_weights);

        let layer_biases: Vec<f64> = layer.biases().iter().map(|&v| &Value v.value()).collect();
        state_dict.insert(bias_key, value: layer_biases);
    }

    state_dict
}
```

Fig1 State dictionary

```
use indexmap::IndexMap;
use serde_pickle::{ser, SerOptions};
use std::fs::File;

impl Model {
    pub fn save_state_dict(&self, path: &str) {
        let mut file: File = File::create(path).unwrap();

        let state_dict: IndexMap<String, Vec<f64>> = self.state_dict();
        micrograd_rs::layers::model::Model
        pub fn state_dict(&self) -> IndexMap<String, Vec<f64>> {
            SerOptions::new().unwrap();
        }
    }
}
```

Fig2 Saving model parameters

```
import pickle

# changes PyTorch generated state dict to micrograd state dict
def convert_state_dict(state_dict):
    new_state_dict = {}
    for name, tensor in state_dict.items():
        new_state_dict[name] = [float(value) for value in tensor.flatten()]
    return new_state_dict

new_state_dict = convert_state_dict(model.state_dict())

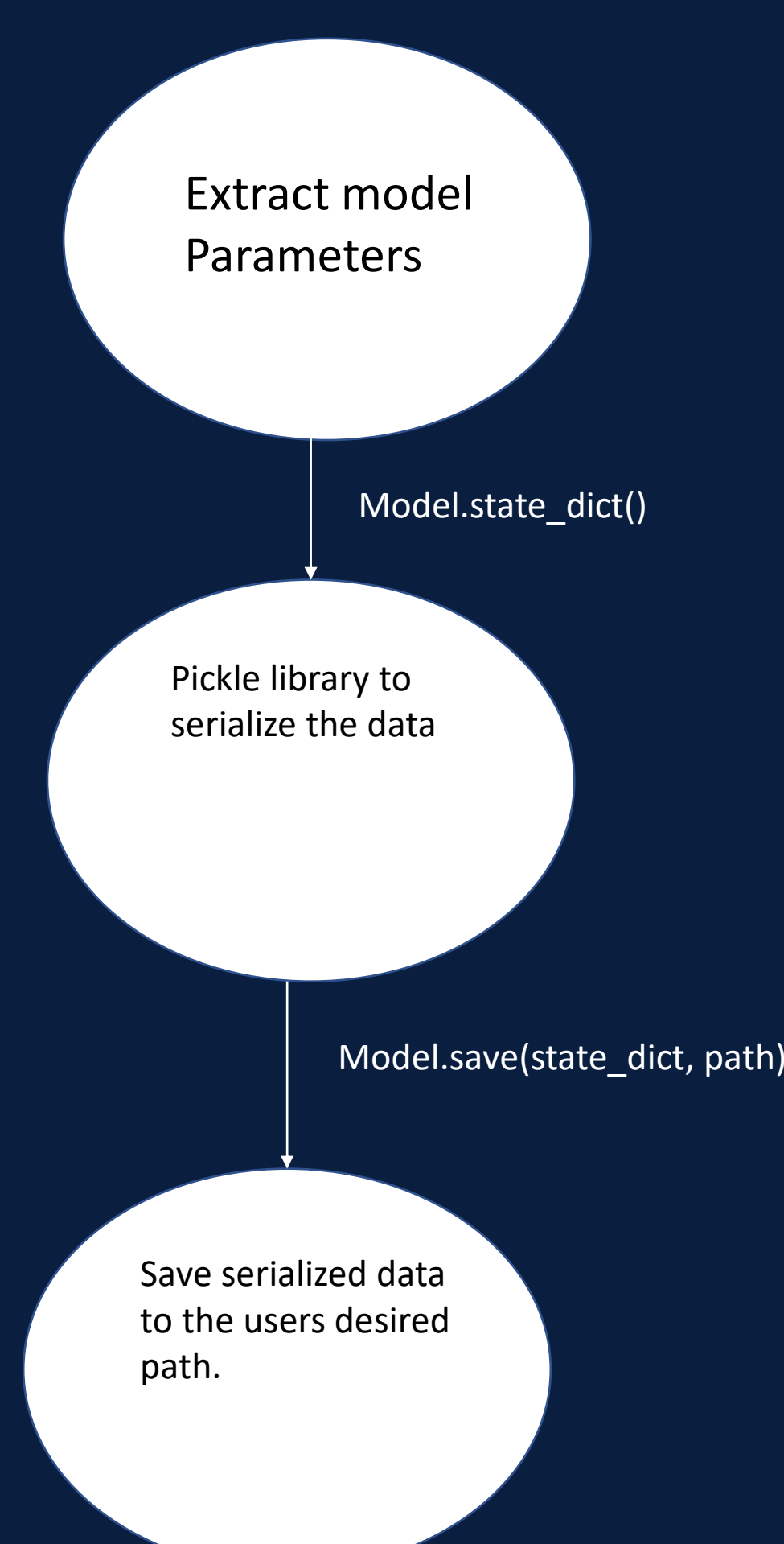
# stores new state dict
with open(path, "wb") as f:
    pickle.dump(new_state_dict, f)
```

Fig3 Saving Pytorch model for compatibility with Micrograd_rs

CONTRIBUTIONS

- **Saving Model Parameters:** Created a function that extracts the model parameters [fig1], and another one that then serializes the parameters and stores it into the user's preferred destination.[Fig2]
- **Pytorch models compatibility with Micrograd_rs Models:** This allows developers to load pytorch models in to Micrograd_rs, providing a more seamless integration [Fig3].
- **Weight Initializer:** Implemented Lecun weight initializer.

SYSTEM DESIGN



SUMMARY

The development of a deep learning framework in Rust combines the strengths of Rust's strong memory safety guarantees and high-performance with deep learning technology. By creating a framework from scratch, We aim to provide an intuitive and optimized tool for building and deploying deep learning models in Rust. This has the potential to enhance the existing deep learning ecosystem and drive innovation in the field of deep learning, especially in domains that require high reliability and safety.

REFERENCES

- Andrej's micrograd: <https://github.com/karpathy/micrograd>
- Pytorch documentation: <https://pytorch.org/docs/stable/nn.htm>
- Micrograd_rs: <https://github.com/LazaroHurtado/micrograd-rs>