ML Kit: A Machine Learning Library for Rust

Owen Wetherbee (ocw6), Ethan Ma (em834), Sylvan Martin (slm338)

Keywords: Machine learning, Gradient Descent, PCA, Rust

Application Setting

Rust is a relatively new programming language with a dearth of machine learning infrastructure. We aimed to create a library for Rust programmers to make machine learning convenient, much like NumPy or TensorFlow in Python.

Project Description

Initially, we wanted to create a comprehensive machine learning library for Rust, which would use Rust's safety and speed to implement many algorithms used in data science. We had the original (lofty) goal of being able to run a diffusion model by the end of the semester, or be able to do anything that NumPy/SciKitLearn could do. As we began implementation we recognized that we lacked the time and resources to implement the sheer amount of algorithms we set out to. So, we shifted our focus to what we believe are some of the core algorithms in the field of Machine Learning.

In the end, we created the pure-Rust library `ml_kit`, which implements, from scratch, the following:

- Neural Network based learning, consisting of
  - the basic neural network model with user-defined network shape and activation functions for each layer
  - functionality for handling large datasets for training and testing
  - a stochastic gradient descent trainer, with whatever batch size and epochs a user may want
  - various "gradient update schedules," such as fixed learning rates, time-decay learning rates, AdaGrad, etc.
  - Convolutional Neural Networks (Owen/Ethan talk more on this?)

- Principle Component Analysis, consisting of
  - an implementation of Singular Value Decomposition (SVD),
  - using SVD to obtain a k-dimensional plane of best fit for a set of points in $\mathbb{R}^n$,
  - compressing images (or any data) using SVD by truncating low-variance dimensions

Neural Networks

Principle Component Analysis and SVD

Singular Value Decomposition (SVD) is a factorization of a matrix $A \in \mathbb{R}^{m\times n}$ into

$$A = U\Sigma V^\top$$

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices with columns $\vec{\mathbf{u}}_1, \dots \vec{\mathbf{u}}_m$ and $\vec{\mathbf{v}}_1, \dots, \vec{\mathbf{v}}_n$ respectively, and Σ is a diagonal matrix of the form (assuming $n \leq m$)

$$\Sigma = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \\ & & & \\ & & & \end{bmatrix} \in \mathbb{R}^{m\times n}$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ are the *singular values* of A. This allows us to write A as the linear combination

$$A = \sum_{i=1}^{n} \sigma_i \vec{\mathbf{u}}_i \vec{\mathbf{v}}_i^\top$$

Note that because the singular values are sorted in decreasing order, we can effectively "save data" in representing A by truncating all but the first $r \leq n$ singular vectors, as the last items in the sum do not contribute as much to the overall product. The immediate application is identifying the most significant axes of correlation in data, allowing dimensionality reduction of datasets by re-writing each item in the basis $\vec{\mathbf{v}}_1, \dots, \vec{\mathbf{v}}_r$. Commonly, the "data" of concern is written into the rows of A.

What we implemented

Our library implements the Golub-Kahan SVD algorithm (described in "Matrix Computations," by Golub and van Loan. [1]) which begins by bi-diagonalizing A, then performing SVD on the bidiagonalization, as the numerical stability and performance are better. Once we were able to compute the SVD of any matrix, we could use SVD as a subroutine in other useful techniques.

The (subjectively) coolest application of SVD we implemented is image compression. By taking an $m \times n$ image and writing it as a 4-tuple of matrices (R, G, B, A) representing the red, green, blue, and alpha channels of the pixels, we can perform SVD on each color channel, store only the SVD representation after truncating insignificant singular vectors, and then de-compress the image later by computing $U\Sigma V^\top$ for each color channel. In practice, one can discard roughly half the singular values and still obtain a recognizeable image. Examples of this and discussion of runtime and compression rates will be left to the evaluation section.

A common goal in statistics is finding the "line of best fit" of a set of points, or in higher dimensions, a k-dimensional plane of best fit. For a cluster of data centered at the origin, the first singular vector, $\vec{\mathbf{v}}_1$, is the line of best fit. This is because SVD is equivalently defined as an optimization proceedure where

$$\vec{\mathbf{v}}_1 = \operatorname*{argmax}_{\|x\|=1} \|Ax\|$$

Since $\vec{\mathbf{v}}_1$ is maximizing the sum of squares of the inner products with all the data points, it is minimizing the sum of squared distances from each data point to the line spanned by $\vec{\mathbf{v}}_1$. More generally, taking the first $\vec{\mathbf{v}}_1, \ldots, \vec{\mathbf{v}}_k$ vectors, we obtain a basis for a k-dimensional "plane of best fit."

We implement a proceedure which takes a data matrix $D \in \mathbb{R}^{n \times m}$ whose columns are each $\mathbb{R}^n$ data points, and returns $\vec{\mu} \in \mathbb{R}^n$ and $V \in \mathbb{R}^{n \times k}$ such that the plane defined by

$$\left\{ \vec{\mu} + \sum_{i=1}^{k} \alpha_i \vec{\mathbf{v}}_i \mid \alpha_i \in \mathbb{R} \right\}$$

is the plane of best fit for the data. Examples of this will be left to the evaluations section.

Relationship to Other Work

Linear algebra is the foundatin to machine learning, so we needed to use a good linear algebra library. Sylvan had previously spent winter break working on `matrix_kit`,

1. Gene H. Golub and Charles F. van Loan, *Matrix Computations*, Fourth (JHU Press, 2013), ISBN: 1421407949 9781421407944, http://www.cs.cornell.edu/cv/GVL4/golubandvanloan.htm.

which is a pure-Rust linear algebra package that implemented incredibly basic matrix-vector operations. We continued developing this library in parallel with `ml_kit` over the semester as we recognized more features that were needed from the linear algebra library. So, the sum of our work for the course can be thought of as the entirety of `ml_kit`, as well as significant improvement to the functionality of `matrix_kit`.

The `matrix_kit` library can be found on GitHub at https://github.com/SylvanM/matrix_kit.

Evaluation

Gradient Descent

Principle Component Analysis

In this section, we discuss the performance of our SVD-based algorithms in terms of their speed and correctness.

Singular Value Decomposition

In numerical methods such as SVD, the accumulation of floating point error prevents us from measuring exact correctness. Instead, we define some small ε, and consider two matrices $A \approx_\varepsilon B$ to be "basically equal" if for all i, j, $|a_{ij} - b_{ij}| < \varepsilon$. When testing on large matrices, we chose $\varepsilon = 1 \times 10^{-5}$ to be our goal, though in practice on smaller matrices ($m, n \leq 50$) we attain correctness up to $\varepsilon = 1 \times 10^{-9}$.

To test correctness of our SVD implementation, we start with a randomly generated matrix $A \in \mathbb{R}^{m \times n}$ where each entry is sampled from the normal distribution $\mathcal{N}(0, 1)$, and compute its SVD into U, V, and Σ. We then check that U and V are orthogonal by testing $U^\top U \approx_\varepsilon I_m$ and $V^\top V \approx_\varepsilon I_n$. We then check that $A \approx_\varepsilon = U \Sigma V^\top$. One "trial" is the above proceedure with m, n sampled uniformly at random from the range 2 to 200. We ran one thousand trials total

Image Compression

It works! Here is an example!

Planes of Best Fit

References

Golub, Gene H., and Charles F. van Loan. *Matrix Computations*. Fourth. JHU Press, 2013. ISBN: 1421407949 9781421407944. http://www.cs.cornell.edu/cv/GVL4/golubandvanloan.htm.