SolMATe: A Linear Algebra Package for Cloud-Based Machine Learning via Blockchain, with Applications in IoT for Web 3.0 and Quantum Computing in the Metaverse

Naveen Arunachalam

School of Engineering
MIT
77 Massachusetts Ave, Cambridge MA 02139
narunach@mit.edu

Abstract

Blockchain technology has the potential to bring disruption to a wide range of disciplines by providing new means of storing, accessing, and processing data. In particular, machine learning is a field that stands to benefit from the accountability, immutability, and reproducibility properties offered by blockchain virtual machines. In this paper, we present the design and implementation of SolMATe, a matrix library for Solidity that enables machine learning directly within Ethereum smart contracts, thus preventing the need for an off-chain oracle for model training and execution. We then use SolMATe to implement both an end-to-end machine learning pipeline and a simulated quantum computer entirely within the Ethereum virtual machine. Future work and implications for IoT devices such as virtual reality headsets are then discussed.

1. Introduction and Approach

Blockchain technology has been of interest since the introduction of Bitcoin in 2008 [4]. Public blockchains provide a wide range of desirable properties for peer-to-peer transactions such as financial transactions, including accountability, immutability, and reproducibility.

In 2015, Ethereum provided a means by which the blockchain can be used to record transactions involving general computation on a virtual machine whose rules of operation are shared by all nodes within the same network [1]. Thus, Ethereum has become a platform for managing a wide range of transactions involving general computation, notably including non-fungible tokens (NFTs).

Within Ethereum, users can deploy a special type of object called a smart contract, which is a piece of code that can be queried by other users on the network via an API. Users can send data and/or tokens to the smart contract and receive data and/or tokens in return. Currently, such smart contracts are typically written in Solidity, which supports basic math operations on the Ethereum Virtual Machine, but has no support for vectors or matrices. Nevertheless, Solidity has been successfully used for advanced computing applications; for example, computational chemistry experiments have been performed directly on the Ethereum virtual machine, as shown in [3].

The lack of matrix operations in Solidity means that certain areas of computation, such as machine learning and scientific computing, would be difficult or impossible to implement within a smart contract. To address this shortcoming, we present the Solidity Matrix Environment (SolMATe), which provides a means for performing matrix operations within an Ethereum smart contract. As demonstrated in this paper, SolMATe can be applied to a wide range of areas, including Internet of Things (IoT), artificial intelligence (AI), big data, cloud computing, Web 3.0, quantum computing, NFTs, and the metaverse.

Broadly, SolMATe is used for performing gas-efficient matrix operations on inputs provided to smart contracts written in Solidity v0.8.11 or above. The main features of SolMATe include:

- Gas-efficient operations with signed 59.18-decimal fixed-point numbers
- Functions for manipulating vectors and matrices
- Linear algebra routines enabling the implementation of machine learning algorithms, such as multivariate linear regression and deep neural networks

SolMATe was designed to add methods to Solidity's base types such as `int256`, `int256[]`, and `int256[][]`. These methods range from array manipulation routines such as transposing (`T`) and array slicing (`sliceVector`, `sliceMatrix`) to matrix utilities such as matrix multiplication (`dot`), eigenvalue solving (`eigenvalues`), and QR decomposition via Householder reflections.

SolMATe can be used in a smart contract by importing `PRBMathSD59x18.sol`, `VectorUtils.sol`, and `MatrixUtils.sol`; the template in 1 demonstrates how this can be performed in Solidity.

2. Example: Machine Learning Pipeline Implemented within a Smart Contract

Here, we provide an example machine pipeline learning implemented directly in a smart contract. The corresponding code for this demonstration can be found at [5].

To set up a machine learning pipeline in SolMATe, it is necessary to (1) define a training data set, (2) specify a mathe-

Copyright © ACM [to be supplied]. . . $15.00
DOI: http://dx.doi.org/10.1145/(to come)

```
// SPDX-License-Identifier: BSD-4-Clause
pragma solidity >=0.8.4;
import "prb-math/contracts/PRBMathSD59x18.sol";
import "./VectorUtils.sol";
import "./MatrixUtils.sol";

contract SolMATe_demo {
    using PRBMathSD59x18 for int256;
    using VectorUtils for int256[];
    using MatrixUtils for int256[][];
    // Code goes here
}
```

Listing 1. Template for SolMATe Smart Contract

matical model and/or solver, and (3) train the model. A training data set can be provided to SolMATe via a contract function (`multipleLinearRegression` in the provided example). The contract function is responsible for receiving training data as an input, selecting a solver, and running a training sequence to produce an output. Training data can be provided in the form of vectors or matrices of type `int256`. Because the type of the input must be `int256`, inputs containing fractional numbers should be represented directly as 59x18 signed decimal numbers. In the example contract, input is provided as integers and then converted to 59x18 signed floating point numbers upon receipt. The two inputs, a coefficient matrix A and a solution vector b, are then used to solve for a vector x that solves the matrix equation $Ax = b$. First, a QR decomposition of A is obtained to derive an orthogonal matrix Q and an upper triangular matrix R. SolMATe provides two different ways of performing a QR decomposition: `QRDecompositionGramSchmidt` and `QRDecompositionHouseholder`, which respectively use a Gram-Schmidt process and Householder reflections; the example contract uses Householder reflections.

From there, the following expression is obtained:

$$A = QR; Ax = b$$

$$QRx = b$$

$$Rx = Q^{-1}b$$

$$Rx = Q^T b$$

$$x = R^{-1}(Q^T b)$$

The pseudocode implmenting the above process is provided below:

Algorithm 1 Pseudocode for Implemented Smart Contract

```
Data: Matrix $A$, vector $b$
Result: $x$ s.t. $Ax = b$
1 Convert $A$ to 59x18 float
2 $Q, R \leftarrow$ QRDecompositionHouseholder($A$)
3 return backSubstitute($R, Q^T b$)
```

As seen above, to solve the equation $Ax = b$, the example contract uses back-substitution (represented by the SolMATe function `backSubstitute`) by recursively solving for the elements of x using the upper triangular representation R.

3. Example: Simulation of a Quantum Computer within a Smart Contract

SolMATe can be used to simulate a rudimentary quantum computer. Here, we provide a demonstration of a quantum circuit involving two qubits and two gates: the Hadamard gate and the controlled not (CNOT) gate. This circuit transforms two input qubits into the Bell state $|\Phi^+\rangle$. We initialize the system using two qubits $x = |00\rangle$, which have the matrix representation

$$x = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}$$

The Hadamard gate is represented by

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}$$

which we then apply to the first qubit only. This expression is represented as $H \otimes I$, which is a 4x4 matrix whose entries are those of H each multiplied by I. Then, we apply the CNOT gate to both qubits, which is represented as

$$CNOT = \begin{pmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{pmatrix}.$$

The resultant quantum circuit is then represented by

$$CNOT(H \otimes I) = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 0 & 1 & 0\\0 & 1 & 0 & 1\\0 & 1 & 0 & -1\\1 & 0 & -1 & 0\end{pmatrix}.$$

When applied to the input vector x, we obtain the Bell state

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\0\\0\\1\end{pmatrix}$$

Within Solidity, this quantum circuit can be represented as a pure function involving the gate matrix $CNOT(H \otimes I)$, which is then multiplied by the input state vector x. The corresponding implementation of this circuit within a smart contract can be found at [6].

4. Discussion

As shown in the previous examples, SolMATe can be used for a variety of tasks spanning diverse domains. By introducing the ability to manipulate and work with matrices within a smart contract, it enables users to submit data to a network of computers running the Ethereum virtual machine and perform distributed, verifiable calculations. This yields some advantages over traditional cloud computing, wherein a user submits data or commands to a self-hosted or off-site computer cluster. For example, a user working with smart contracts for distributed computing is able to take advantage of computational resources not directly owned by the user, yet still obtain trustworthy results. In the case of machine learning models deployed to the cloud, users often would like to know the data and training pipelines behind the models they are using. Smart contracts implemented with SolMATe provide a way of addressing these concerns by logging data input and model training transactions directly to the virtual ledger.

SolMATe smart contracts have a wide range of practical applications. Second-generation Internet services (i.e. Web 2.0 services) can query SolMATe smart contracts via the an Ethereum API such as the geth API [2]. In the case of embedded systems or IoT devices, significant machine learning workloads can be outsourced to the blockchain via an API query, freeing onsite computing resources. Third-generation Internet services (i.e. Web 3.0 services) also stand to benefit from the use of machine learning within smart contracts rather than in an off-chain oracle because of the auditability properties of an on-chain solution.

As shown in the second example, SolMATe can be used for quantum computing applications. Because SolMATTe contracts can interoperate with Web 2.0 and Web 3.0 seamlessly, it is an ideal development environment for interfacing virtual reality (VR) applications with the blockchain. VR environments running on hardware headsets often use matrix data in the form of positions, orientations, and display data, and this information can be sent to a smart contract implemented in SolMATe to free up resources on the host device.

5. Conclusion and Future Directions

SolMATe provides an environment in which Ethereum smart contracts can perform matrix and array operations directly on chain. It should be noted that operations performed by SolMATe involve high gas costs, making the package impractical for Layer 1 applications; however, it is suitable for Layer 2 applications. In order to mitigate gas costs, future work will focus on making operations within SolMATe more gas-efficient.

By using the methods provided in SolMATe, we present the first instances of end-to-end machine learning and simulation of a quantum computer ever performed on the Ethereum blockchain. These examples illustrate the power and flexibility of the library, opening up many new directions for the applicability of smart contracts. Future projects, such as the Solidity Image Processor (SIMP), aim to use the utilities provided by SolMATe to handle image data, which can then be used to process image-based non-fungible tokens directly on the blockchain.

References

[1] Vitalik Buterin. Ethereum white paper: A next generation smart contract & decentralized application platform. 2013.

[2] Go Ethereum Project. Go Ethereum API Specification. https://geth.ethereum.org/docs/rpc/server, Dec 2021.

[3] Magnus W. Hanson-Heine and Alexander P. Ashmore. Computational chemistry experiments performed directly on a blockchain virtual computer. *Chemical Science*, 11(18):4644–4647, 2020.

[4] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[5] NTA-Capital. SolMATe ML demo. https://github.com/NTA-Capital/SolMATe/blob/main/contracts/SolMATe_demo.sol, Dec 2021.

[6] NTA-Capital. SolMATe quantum computer demo. https://github.com/NTA-Capital/SolMATe/blob/main/contracts/SolMATe_quantum.sol, Dec 2021.