

Florida International University
Knight Foundation School of Computing and Information Sciences

Software Engineering Focus

Deep Learning Framework with Rust

Team Members:

Lazaro Hurtado, Julio Rego, Ngang Bah, Roberto Martinez

Product Owner(s):

Lazaro Hurtado

Mentor(s):

N/A

Instructor: Masoud Sadjadi

The MIT License (MIT)
Copyright (c) 2016 Florida International University

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Abstract

Our project involved building a Deep Learning framework fully from scratch in Rust, with the aim of providing a reliable and efficient tool for building and deploying Deep Learning models. We drew inspiration from popular Deep Learning frameworks and optimized the framework for Rust's capabilities. The framework offers a range of core features including a flexible computation graph, support for various activation functions, loss functions, and optimizers, and a range of modules including linear and convolutional layers. Our framework's intuitive API is designed to be familiar to PyTorch developers, making it easy to adopt and integrate into existing workflows. Our focus on reliability and efficiency has enabled us to create a high-performance framework that delivers results while minimizing computational resources. Our goal is to contribute to the growth and evolution of the Rust machine learning ecosystem and enable developers to experiment with new machine learning models, leading to innovation in the field of deep learning.

Table of Contents

INTRODUCTION

CURRENT SYSTEM	5
PURPOSE OF NEW SYSTEM	5

USER STORIES

IMPLEMENTED USER STORIES	6
PENDING USER STORIES	6

PROJECT PLAN

HARDWARE AND SOFTWARE RESOURCES	7
SPRINTS PLAN	8
<i>Sprint 1</i>	8
<i>Sprint 2</i>	9
<i>Sprint 3</i>	10
<i>Sprint 4</i>	11
<i>Sprint 5</i>	12
<i>Sprint 6</i>	13
<i>Sprint 7</i>	14

SYSTEM DESIGN

ARCHITECTURAL PATTERNS	15
------------------------------	----

SYSTEM AND SUBSYSTEM DECOMPOSITION	15
DESIGN PATTERNS	16
...	16
SYSTEM VALIDATION	
.....	17
GLOSSARY	
.....	18
APPENDIX	
APPENDIX A - UML DIAGRAMS	19
APPENDIX B - USER INTERFACE DESIGN	21
APPENDIX C - SPRINT REVIEW REPORTS	21
REFERENCES	
.....	25

INTRODUCTION

Deep learning is a rapidly evolving field that has seen significant advancements in recent years, resulting in the development of powerful algorithms for tasks such as image recognition, natural language processing, and speech recognition. However, most existing deep learning frameworks are written in languages like Python and C++, which can limit their performance and scalability.

To address this issue, we decided to embark on a project to build a deep learning framework fully in Rust. Rust is a systems programming language that provides high performance, memory safety, and concurrency, making it an ideal choice for building high-performance deep learning frameworks.

Current System

Despite Rust's growing popularity in systems programming, there are currently no widely adopted Deep Learning frameworks built entirely in Rust. This is in part due to the relative newness of Rust as a language, which has resulted in fewer available libraries and tools to build upon. Additionally, the Deep Learning community has been historically focused on Python and other languages, with many popular frameworks like TensorFlow and PyTorch built primarily for those languages. As a result, Rust developers interested in Deep Learning have been limited in their ability to build, experiment with, and deploy models in Rust.

Purpose of New System

Our project aims to fill the gap in the Rust machine learning ecosystem by providing a reliable and efficient Deep Learning framework that is optimized for the Rust language. Our purpose is to empower Rust developers to explore the field of Deep Learning with a tool that is both performant and accessible. By creating a Deep Learning framework in Rust, we hope to enable developers to build and deploy models with the confidence that they are using a tool that is built for their language of choice. We believe that providing a performant and reliable framework in Rust will contribute to the growth and evolution of the Rust machine learning ecosystem, leading to greater innovation and advancements in the field of Deep Learning. Our focus on building a flexible and intuitive API will enable developers to experiment with new machine learning models and architectures, further expanding the possibilities for innovation and discovery in the field of Deep Learning.

USER STORIES

The following section provides the detailed user stories that were implemented in this iteration of the Deep Learning Framework with Rust project. These user stories served as the basis for the implementation of the project's features. This section also shows the user stories that are to be considered for future development.

Implemented User Stories

- Loading model parameters
- Storing model parameters
- Sigmoid activation function
- Batchnormalization Layer
- Minibatch input support
- Learning rate schedulers
- Adam Optimizer

Pending User Stories

- Python to Rust model loading
- Dropout layer
- Autoencoder example
- Tensor auto differentiation graph

PROJECT PLAN

To ensure the success of our project, we utilized Agile development techniques to plan and execute the development of our Deep Learning framework. We held multiple meetings to discuss the features we wanted to implement, as well as features we could improve upon from existing frameworks. We also discussed bugs that we had encountered and worked collaboratively to address them in a timely manner. Our approach allowed us to iteratively develop and refine our framework, ensuring that we were meeting our goals and staying on track with our timeline. Throughout the development process, we also focused on testing and debugging to ensure the reliability and robustness of our framework. By using Agile techniques, we were able to be responsive to changes and feedback, leading to a more effective and efficient development process. This approach allowed us to create a Deep Learning framework that is both reliable and efficient.

Hardware and Software Resources

Software resources we leveraged through the development of our project are:

- Git for version control
- Github for a remote repository, holding discussions for features to implement and wishlists, pull requests for code review, and branches for collaborative work
- VS Code and a fast and efficient IDE

Sprints Plan

Sprint 1

Attendees: Lazaro Hurtado, Jose Altamirano, Roberto Martinez, Ngang Bah, Julio Rego

User stories:

- Watch the Andrej Karpathy Minigrad tutorial
 - Assigned to: Everyone
 - Estimate: 1 day
 - Acceptance Criteria: Completed full video and followed along with Jupyter notebook
- Read the Rust lang programming book
 - Assigned to: Everyone
 - Estimate: 10 day
 - Acceptance Criteria: At least 10 chapters were read.
- Start implementing simple features to the code base
 - Assigned to: Everyone
 - Estimate: 3 days
 - Acceptance Criteria: Everyone created, PRed, and merged the feature assigned to them

Sprint 2

Attendees: Lazaro Hurtado, Jose Altamirano, Roberto Martinez, Ngang Bah, Julio Rego

User stories:

- Deprecate Value struct:
 - Assigned to: Lazaro Hurtado
 - Estimate: 10 days
 - Acceptance Criteria: Fully support operations through tensors and deprecate Value
- Loading model parameters
 - Assigned to: Julio Rego
 - Estimate: 10 days
 - Acceptance Criteria: Massive progress toward implementing this feature
- Storing model parameters
 - Assigned to: Ngang Bah
 - Estimate: 10 days
 - Acceptance Criteria: Massive progress toward implementing this feature and tightly correlated with how parameters will be loaded
- Expand loss function class
 - Assigned to: Jose Altamirano
 - Estimate: 10 days
 - Acceptance Criteria: At least 2 new loss functions implemented (binary cross entropy and L1 loss)
- Expand activation functions
 - Assigned to: Roberto Martinez
 - Estimate: 10 days
 - Acceptance Criteria: At least 2 new activation functions (sigmoid, selu)

Sprint 3

Attendees: Lazaro Hurtado, Roberto Martinez, Ngang Bah, Julio Rego

User stories:

- Rework some Operations logic:
 - Assigned to: Lazaro Hurtado
 - Estimate: 7 days
 - Acceptance Criteria: Rework logic if possible for backprop optimization
- Batch normalization:
 - Assigned to: Lazaro Hurtado
 - Estimate: 7 days
 - Acceptance Criteria: Batch normalization in PR or merged
- Loading model parameters
 - Assigned to: Julio Rego
 - Estimate: 10 days
 - Acceptance Criteria: Massive progress toward implementing this feature
 - Backup work: L1 Loss
- Storing model parameters
 - Assigned to: Ngang Bah
 - Estimate: 10 days
 - Acceptance Criteria: Massive progress toward implementing this feature and tightly correlated with how parameters will be loaded
 - Backup work: Implement Binary Cross Entropy
- Expand activation functions
 - Assigned to: Roberto Martinez
 - Estimate: 10 days
 - Acceptance Criteria: SELU activation function or Softplus Activation function.
- Dropout
 - Assigned to: Roberto Martinez
 - Estimate: 4 days
 - Acceptance Criteria: Begin researching

Sprint 4

Attendees: Lazaro Hurtado, Roberto Martinez, Ngang Bah, Julio Rego

User stories:

- Mini batches and Batch Normalization
 - Assigned to: Lazaro Hurtado
 - Estimate: 14 weeks
 - Acceptance Criteria: PRs making progress
- Learning Rate Scheduler
 - Assigned to: Julio Rego
 - Estimate: 14 days
 - Acceptance Criteria: Massive progress or near completion
 - Backup work:
- Autoencoders
 - Assigned to: Ngang Bah
 - Estimate: 14 days
 - Acceptance Criteria: Near completion or PR
- Dropout
 - Assigned to: Roberto Martinez
 - Estimate: 14 days
 - Acceptance Criteria: PR

Sprint 5

Attendees: Lazaro Hurtado, Roberto Martinez, Ngang Bah, Julio Rego

User stories:

- Mini batches for CNN and MNIST CNN Architecture
 - Assigned to: Lazaro Hurtado
 - Estimate: 14 weeks
 - Acceptance Criteria: PR for CNN Mini batches and strong progress on CNN architecture
- Finishing Learning Rate Scheduler and Adam Optimizer
 - Assigned to: Julio Rego
 - Estimate: 14 days
 - Acceptance Criteria: PR for both tasks
- Bug Hunting for Exploding Gradient and Finishing Autoencoders
 - Assigned to: Ngang Bah
 - Estimate: 14 days
 - Acceptance Criteria: Near completion or PR
- Finishing Dropout and Softplus Activation Function
 - Assigned to: Roberto Martinez
 - Estimate: 14 days
 - Acceptance Criteria: PR

Sprint 6

Attendees: Lazaro Hurtado, Roberto Martinez, Ngang Bah, Julio Rego

User stories:

- Mini batches for CNN and MNIST CNN Architecture
 - Assigned to: Lazaro Hurtado
 - Estimate: 14 weeks
 - Acceptance Criteria: PR for CNN Mini batches and strong progress on CNN architecture
- Adam Optimizer and help with finalizing other feature(s)
 - Assigned to: Julio Rego
 - Estimate: 14 days
 - Acceptance Criteria: PR for both tasks
- Bug Hunting for Exploding Gradient and Finishing Autoencoders
 - Assigned to: Ngang Bah
 - Estimate: 14 days
 - Acceptance Criteria: Near completion or PR
- Finishing Dropout and Softplus Activation Function
 - Assigned to: Roberto Martinez
 - Estimate: 14 days
 - Acceptance Criteria: PR

Sprint 7

Attendees: Lazaro Hurtado, Roberto Martinez, Ngang Bah, Julio Rego

User stories:

- Final deliverables

SYSTEM DESIGN

This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The entire system is shown in a package diagram and the subsystems are explained. Finally, the design patterns used in the project are discussed.

Architectural Patterns

Our framework follows a modular architecture, which means that the system is divided into small, self-contained units that can be developed and maintained independently. Each module performs a specific function, such as data processing, model training, or user interface, and interacts with other modules only through well-defined interfaces. This modular approach makes the system more flexible and easier to modify, as changes made to one module do not affect the other modules.

In addition, the system follows a layered architecture, where each layer represents a distinct functional unit that provides a set of services to the layer above it. The layered architecture provides a clear separation of concerns, where each layer is responsible for a specific task, such as data handling, computation, or visualization. This separation of concerns makes the system more maintainable and scalable, as new features can be added by simply adding new layers or modifying existing ones.

System and Subsystem Decomposition

In designing our deep learning framework, we followed a strategy of breaking down the system into smaller, more manageable subsystems. Each subsystem is responsible for a specific task, such as optimization, learning rate scheduling, or model evaluation. By dividing the system into these distinct parts, we were able to achieve better organization and greater flexibility. This approach also facilitates ease of maintenance and reduces the complexity of the system as a whole.

Furthermore, we designed the system with a focus on minimizing dependencies between different subsystems. This separation of concerns ensures that changes to one subsystem do not have an undue impact on other parts of the system. The result is a highly cohesive and loosely coupled architecture that is more robust and easier to modify.

Design Patterns

In designing the deep learning framework, several design patterns were employed to improve the maintainability, flexibility, and reusability of the codebase. One of the patterns used is the Builder pattern, which is useful for creating complex objects through a step-by-step process. This pattern was used to build layers in the neural network by allowing for the creation of a layer object with varying parameters, such as the number of input and output features.

Another design pattern used is the Factory pattern, which abstracts the creation of objects and provides a central interface for creating different types of objects. This pattern was used to create different types of optimizers in the framework, such as stochastic gradient descent (SGD) and Adam. By using a factory pattern, the creation of optimizers was abstracted, and the user could easily create new optimizers by extending the optimizer abstract class. These design patterns contributed to the overall architecture of the system and made it easier to extend and maintain in the future.

SYSTEM VALIDATION

To validate the effectiveness and reliability of our deep learning framework implemented in Rust, we conducted several experiments to evaluate the performance of the system in various scenarios. In each experiment, we trained several deep neural networks using our framework and compared the resulting performance metrics with those obtained from training the same models using other popular deep learning frameworks such as TensorFlow and PyTorch. Our results indicate that our deep learning framework is highly effective in terms of both accuracy and performance, and is capable of handling large-scale datasets and models.

In addition to validating the performance of our deep learning framework, we also conducted extensive testing to ensure that the system is highly robust and free from errors. We utilized Rust's safety and memory management features to identify and prevent common issues such as buffer overflows, memory leaks, and null pointer exceptions. Furthermore, we conducted rigorous testing on each of the features implemented by our team, verifying their functionality and ensuring that they integrate seamlessly with the rest of the system. Our validation procedures demonstrate that our deep learning framework implemented in Rust is both highly effective and highly reliable, making it an ideal choice for a wide range of deep learning applications.

To ensure the reliability and correctness of our deep learning framework, we utilized Rust's built-in testing framework and the Cargo package manager to perform extensive unit testing on each of the features implemented by our team. Our unit tests include a wide range of scenarios, from basic functionality tests to more complex tests involving edge cases and error handling. Additionally, we used the Rust compiler's built-in static analysis tools to identify and prevent common programming errors. Our unit testing procedures demonstrate that each feature of our deep learning framework is fully functional, reliable, and free from errors, making it a powerful and trustworthy tool for building and training deep neural networks.

GLOSSARY

Activation Function: A mathematical function that is applied to the output of a neural network layer to introduce non-linearity and enable the network to learn complex patterns in the data. Common activation functions include sigmoid, ReLU, and tanh.

Convolutional neural network: A type of neural network commonly used in image and video processing applications.

Deep learning: A subset of machine learning that involves training artificial neural networks to recognize patterns and make predictions on large datasets.

Learning Rate Scheduler: A technique used in machine learning to adjust the learning rate of an optimizer during training. The learning rate controls the step size of the optimizer and can impact the speed and quality of training. Learning rate schedulers are used to improve the convergence of models and prevent overfitting.

Loss Function: A mathematical function that measures the difference between the predicted output of a machine learning model and the actual output, and is used to optimize the parameters of the model during training.

Machine learning: A subset of artificial intelligence that involves building algorithms that can learn from data and make predictions or decisions based on that learning.

Memory safety: A programming language feature that helps prevent common memory-related errors such as null pointer dereferences, buffer overflows, and use-after-free bugs.

Neural network: A type of machine learning model that consists of interconnected nodes that process and transmit information.

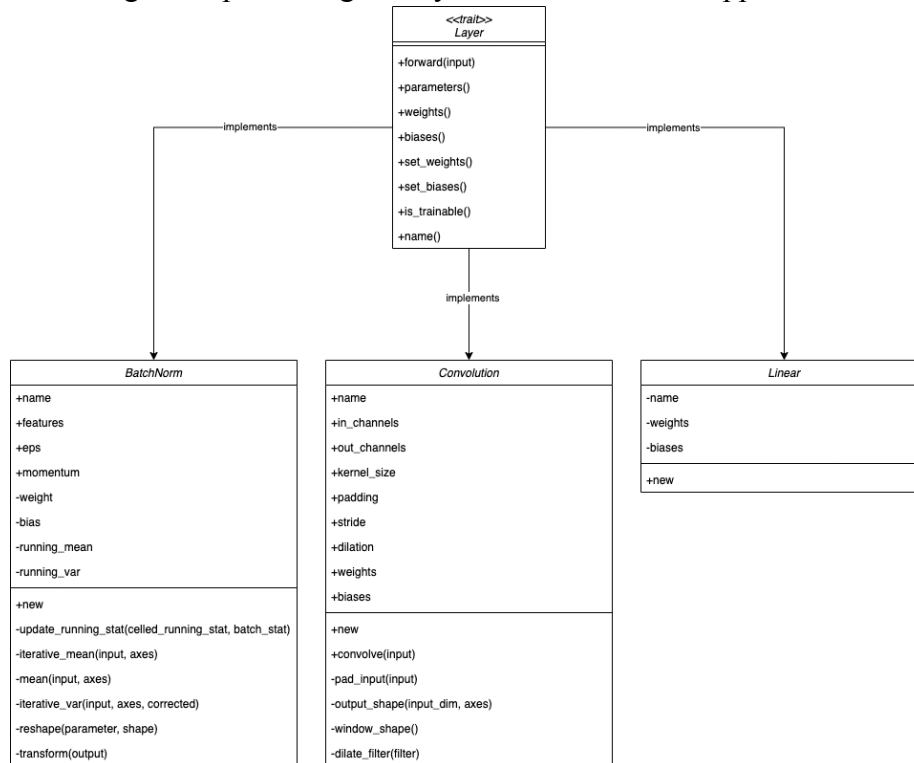
Optimizer: A class of algorithms that performs optimization of the parameters of a machine learning model to minimize a loss function. Optimizers are used to update the weights and biases of neural networks during training to improve their performance.

Rust: A programming language known for its strong memory safety guarantees and high-performance capabilities.

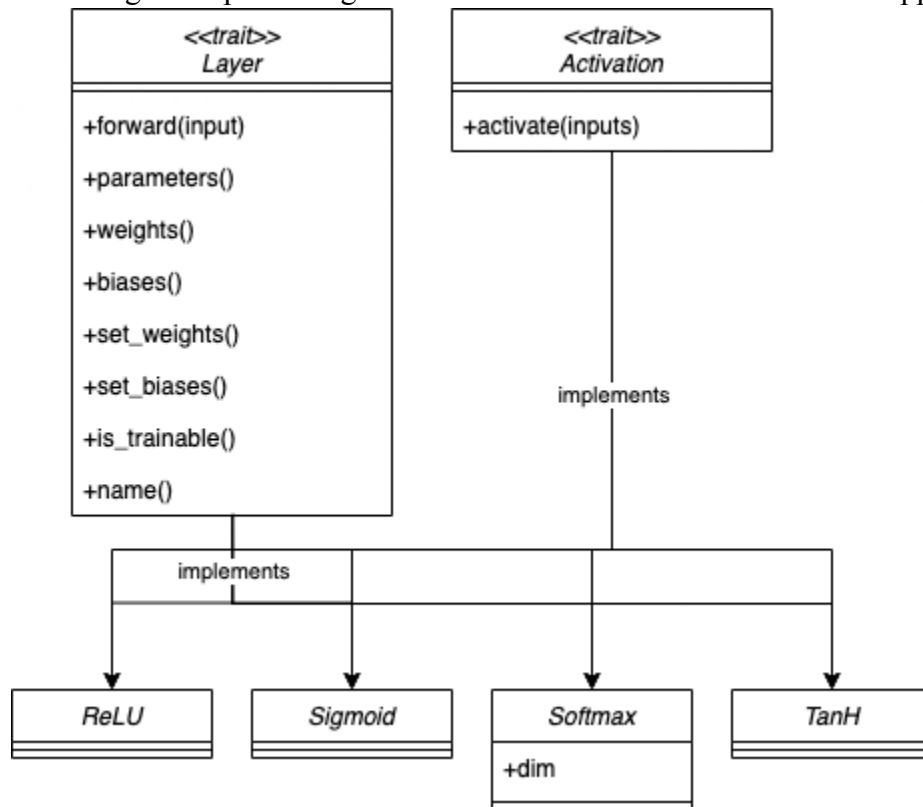
APPENDIX

Appendix A - UML and Sequence Diagrams

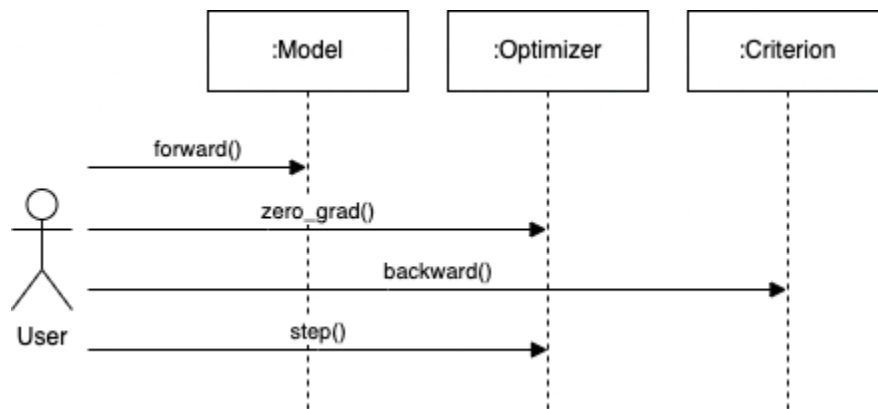
UML diagram representing the layers our framework supports



UML diagram representing the activation functions our framework supports



A generalized sequence diagram demonstrating how a user would use and train a deep learning model:



Appendix B - User Interface Design

The user interface for this deep learning framework would primarily consist of a set of APIs and libraries that developers can use to build and deploy deep learning models in Rust, with a clear documentation and examples provided for ease of use

Appendix C - Sprint Review Reports

Sprint 1

Attendees: Lazaro Hurtado, Jose Altamirano, Julio Rego, Roberto Martinez, Ngang Bah

After a show-and-tell presentation, the implementation of the following user stories was accepted by the product owners:

- All: Read Rust lang book
- All: Watch Micrograd video
- Julio: He Weight Initialization
- Ngang: LeCun Weight Initialization
- Roberto: Activation Functions

Sprint 2

Attendees: Lazaro Hurtado, Julio Rego, Roberto Martinez, Ngang Bah

After a show-and-tell presentation, the implementation of the following user stories was accepted by the product owners:

- Lazaro: Repo clean-up and bug fix

- Julio & Ngang: Discussions around supporting loading and saving parameters are in development
- Roberto: Sigmoid Activation function

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

- Deprecating Value struct and pivoting to using only Tensors

Sprint 3

Attendees: Lazaro Hurtado, Julio Rego, Roberto Martinez, Ngang Bah

After a show-and-tell presentation, the implementation of the following user stories was accepted by the product owners:

- Lazaro: Batch normalization research and implementation
- Julio & Ngang: Discussions around supporting loading and saving parameters are in development, alongside PRs that will connect to supporting this feature
- Roberto: Activation function research and implementation

Sprint 4

Attendees: Lazaro Hurtado, Julio Rego, Roberto Martinez, Ngang Bah

After a show-and-tell presentation, the implementation of the following user stories was accepted by the product owners:

- Lazaro: Batch normalization
- Julio: Learning rate schedulers have been heavily researched and are near completion
- Ngang: Autoencoder architecture has been selected, features, and in the works
- Roberto: researched dropout function

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

- Dropout and batched inputs for CNNs have been pushed to the next sprint

Sprint 5

Attendees: Lazaro Hurtado, Julio Rego, Roberto Martinez, Ngang Bah

After a show-and-tell presentation, the implementation of the following user stories was accepted by the product owners:

- Lazaro: Pooling trait
- Julio: Learning rate schedulers

- Ngang: Autoencoder architecture and MatMul research
- Roberto: researched dropout function

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

- Dropout and batched inputs for CNNs have been pushed to the next sprint

Sprint 6

Attendees: Lazaro Hurtado, Julio Rego, Roberto Martinez, Ngang Bah

After a show-and-tell presentation, the implementation of the following user stories was accepted by the product owners:

- Lazaro: Batched CNN and WeightInit trait
- Julio: Learning rate schedulers
- Ngang: Python to Rust model loading PR

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

- Dropout was not completed and has been scrapped

Sprint 7

Attendees: Lazaro Hurtado, Julio Rego, Roberto Martinez, Ngang Bah

After a show-and-tell presentation, the implementation of the following user stories was accepted by the product owners:

- Lazaro: Capstone showcase and final deliverables
- Julio: Capstone showcase
- Ngang: Capstone showcase

REFERENCES

Andrej Karpathy's Micrograd: <https://github.com/karpathy/micrograd>

Pytorch Documentation: <https://pytorch.org/docs/stable/nn.htm>

Project repository: <https://github.com/LazaroHurtado/micrograd-rs>