System Design Document

For

EasyNN

Dan Wilczak

Jack Nguyen

Liam Kehoe

Nathan Foster

Version/Author	Date
1.0 / Wilczak/Nguyen/Kehoe/Foster	9/28/2021
2.0 / Wilczak/Nguyen/Kehoe/Foster	10/26/2021
3.0 / Wilczak/Nguyen/Kehoe/Foster	11/30/2021

TABLE OF CONTENTS

SYSTEM DESIGN DOCUMENT

1 INTRODUCTION

1.1 Purpose and Scope

The SDD will lay out the general structure for the EasyNN and where responsibilities lay.

1.2 Project Executive Summary

1.2.1 System Overview

EasyNN is a python package designed to provide easy-to-use Neural Networks for users to implement with their own data, with all the features of other large name Neural Network packages. The package is designed to work right out of the box, with many of the common datasets out there. If the datasets have not already been created and trained then we offer a common approach as well. The package is also light-weight, relying mostly on NumPy and not requiring larger packages like SciPy or Jax. Since the scope is much smaller, the framework's design aims to provide much of the common functionalities as other neural network packages without requiring complicated code e.g. GPU acceleration or automatic differentiation.

1.2.2 Design Constraints

The design is constrained by Python, a high-level language known to be less efficient than some other languages. It is also required that the design avoids complicated features not readily provided by NumPY, such as GPU acceleration, to simplify the overall design at the cost of potential computational resources. This will help make the overall project more feasible to manage.

1.2.3 Future Contingencies

EasyNN is designed to be easy for users to work with with very little prior experience; if there are enough complaints of overcomplication, the system will be simplified as needed. In the future, if it is deemed feasible, some constraints may be relaxed. For example, GPU acceleration may be introduced at a later date, or Numba JIT compilation may be used to reach C-like speed.

1.3 Document Organization

Section 2 describes the system architecture/design for the project, and includes a class diagram. Section 3 discusses how users will interact with the EasyNN software. Section 4 goes into gritty detail about how the software operates and was constructed. Section 5 is a brief section about the external interfaces used by EasyNN (currently EasyNN does not utilize external interfaces but the section exists in case that changes in the future). Section 6 describes the security protocols EasyNN takes to preserve user data.

1.4 Project References

EasyGA was a previous Python package designed by some of the same team members.

The EasyGA framework allows users to implement genetic algorithms. EasyNN is a separate project based around neural networks which shares the same vision as EasyGA: providing a framework which is easy for users to use and develop with.

Kinsley, H., & Kukieła, D. (2020). *Neural networks from scratch in Python Building Neural Networks in raw python*. Verlag nicht ermittelbar.

1.5 Glossary

1. Artificial Neural Network (ANN/NN): A simplified computational model of the human brain, loosely based on the idea of passing information between several neurons to get an output.
2. Back Propagation: The process of processing values through a Neural Network, starting from the input nodes and moving to the output nodes.
3. Forward Propagation: The process of processing values through a Neural Network, starting from the input nodes and moving to the output nodes.
4. Loss function: Numerical measurement of the performance of a model.
5. Machine learning: Process of optimizing a model.
6. Model: Representation of another system.
7. Neuron: Component in a Neural Network that collects, stores, and sends values.
8. Optimization: The process of either minimizing or maximizing a loss function.
9. Tensor: Multidimensional array with a shape and indexing by references/pointers. Examples of tensors are floats (except for the indexing), vectors, and matrices. This type is designated as ArrayND e.g. Array1D for a vector and Array2D for a matrix.

2 SYSTEM ARCHITECTURE

2.1 System Hardware Architecture

Since this system is a Python package, there are no physical hardware components to the system. Since GPUs are not used, no special hardware is required either. Computers running EasyNN don’t require more than just 1GB, unless users utilize a large dataset.

2.2 System Software Architecture

The program is modular in nature and broken down into many smaller files that all call and interact with each other, with each doing its own tasks.

The idea is that the model acts as the main interface and therefore houses most of the objects and methods. Within other classes, additional functionality is included, which may be used by other classes. For example, the model's training method runs the optimizer's training method, using itself as the optimizer's model. The model's accuracy/classify methods run its forward method followed by the classifier's accuracy/classify methods. The optimizer uses the datasets on each individual model it is training on.

Each subsystem with further detail is laid out further in Section 4.

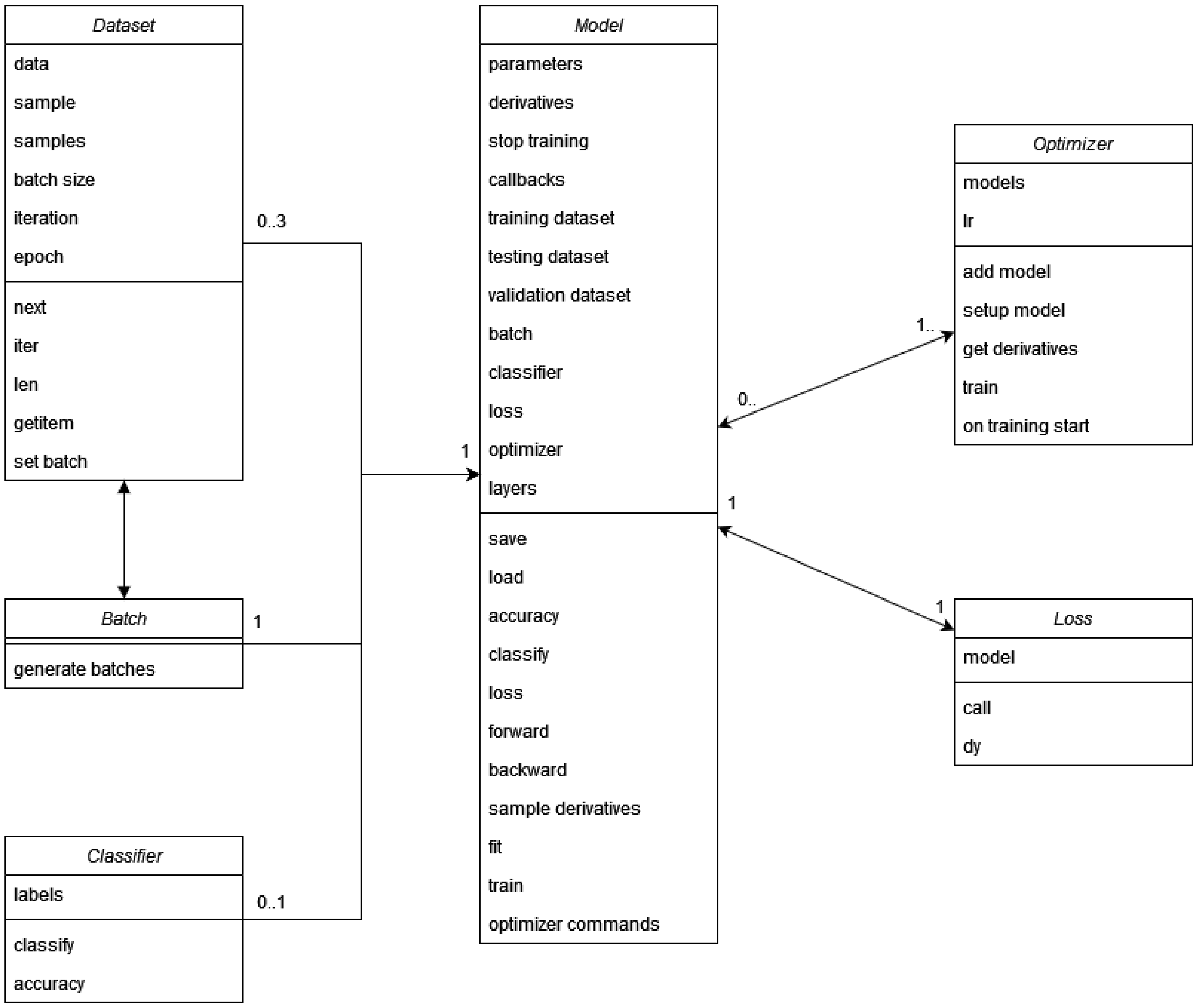

Figure 1: The class diagram.

2.3 Internal Communications Architecture

In this section, describe the overall communications within the system; for example, LANs, buses, etc. Include the communications architecture(s) being implemented, such as X.25, Token Ring, etc. Provide a diagram depicting the communications path(s) between the system and subsystem modules. If appropriate, use subsections to address each architecture being employed.

Note: The diagrams should map to the FRD context diagrams.

No such communication exists in this system, it is a pretty straight forward production/display of information.

3 HUMAN-MACHINE INTERFACE

3.1 Inputs

The user will download the EasyNN package and open the run.py file. In that file they will be able to edit any of the code they desire and then run, or just run the sample data that is preloaded. All the user needs is the path of their image in their system to implement it into the EasyNN model and view the result.

The user may also Pip install the EasyNN package and write their own code using EasyNN in a format similar to the run.py file.

3.2 Outputs

The only output for the system currently are images that are displayed to the user and a prediction of what the network thinks the user input resembles. A sample of the output is on the GitHub wiki section and the readme.

4 DETAILED DESIGN

4.1 Software Detailed Design

The EasyNN file structure is broken down into top-level modules and subdirectories. These top-level modules contain general-use things, such as type-hints which will be used throughout the EasyNN package. Each subdirectory represents a class, whose abstract base class is defined in its abc.py module. Implementations of this abstract base class are then included in separate modules under the same directory, or possibly in further subdirectories as necessary e.g., the model subdirectory contains the activation subdirectory, which contains classes such as ReLU. Each subdirectory should also support a __init__.py module, which controls what should be exported from the subdirectory.

4.2 Internal Communications Detailed Design

During training, the Model, Loss, Optimizer, and Dataset work hand-in-hand to train the Model. Each class should only speak to the next class, as shown in figure 2 below. This enables components to be swapped out e.g. a Model-free Loss implementation can be done provided forward/backward methods and parameters/derivatives i.e. a provided differentiable function.

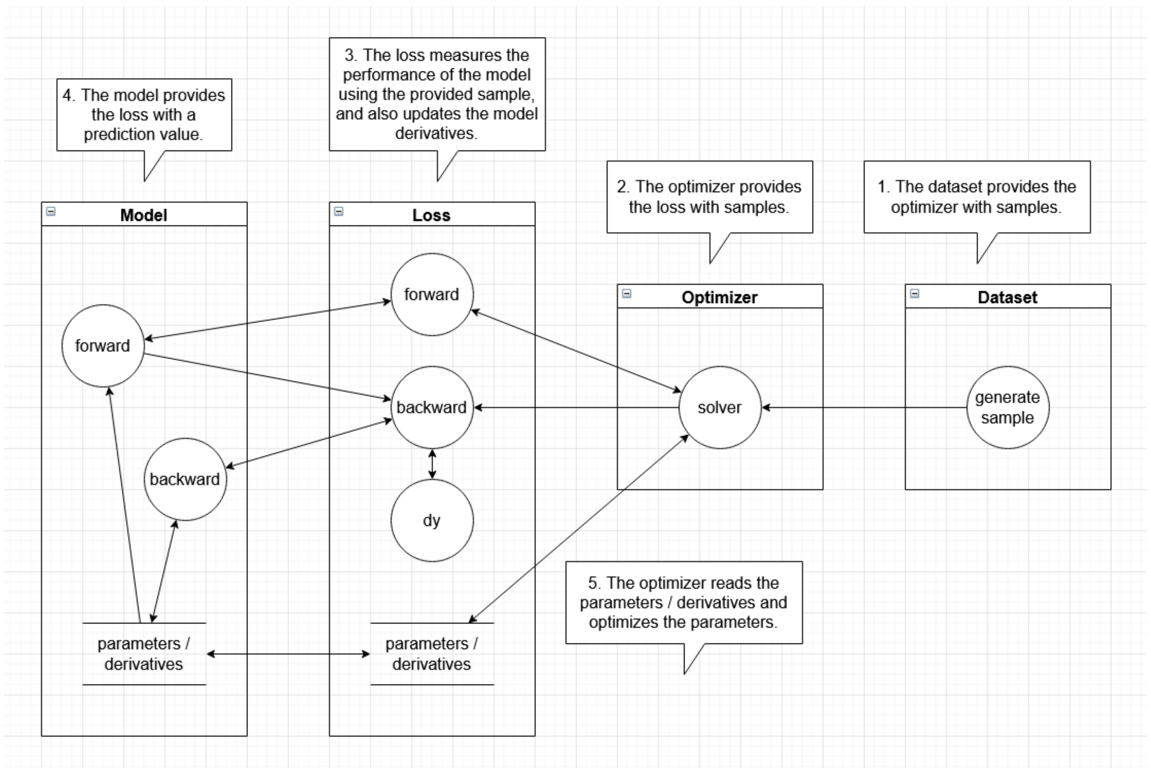
4. The model provides the loss with a prediction value.
3. The loss measures the performance of the model using the provided sample, and also updates the model derivatives.
2. The optimizer provides the loss with samples.
1. The dataset provides the optimizer with samples.
Model
Loss
Optimizer
Dataset
forward
backward
forward
backward
dy
solver
generate sample
parameters / derivatives
parameters / derivatives
5. The optimizer reads the parameters / derivatives and optimizes the parameters.

Figure 2: The training procedure.

In order for the dataset to generate samples as shown in figure 1, a batch is used to generate samples from a dataset. This is shown in figure 2. The separation of the dataset and batch allows the batch to generate samples for multiple datasets, as demonstrated in figure 3. This makes sense because there is usually a training dataset and a testing dataset which need to generate samples concurrently.

Figures 4 through 7 show the general forward/backward propagation procedures. Models shall take in inputs and return outputs by using their parameters. They will then be able to take derivatives and return derivatives, as well as update the derivatives of their own parameters. Furthermore, they shall support intra-parallelism by accepting multiple inputs simultaneously. This allows matrix-based-parallelism to be used, which takes advantage of cache efficiencies and sometimes special algorithms handled by NumPy.

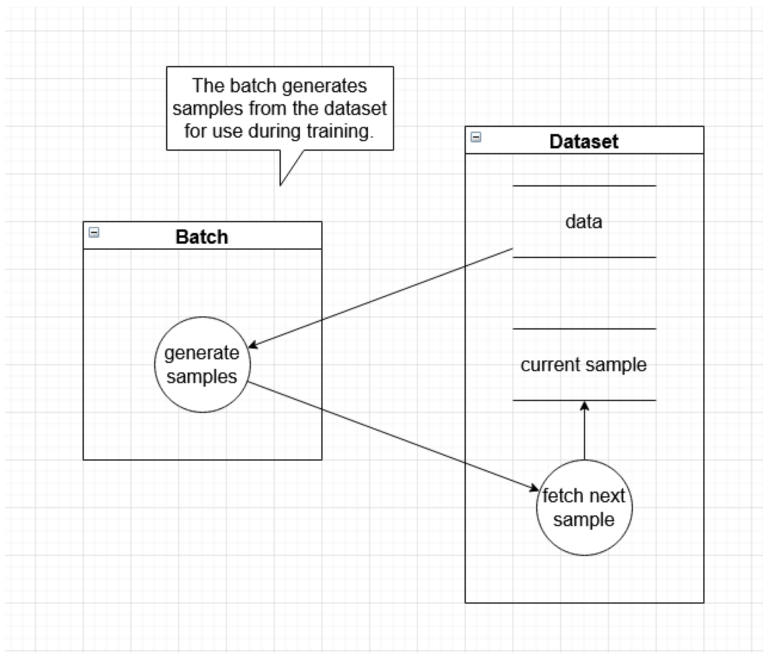
The batch generates samples from the dataset for use during training.
Batch
generate samples
Dataset
data
current sample
fetch next sample

Figure 3: Generating samples from the Dataset using the Batch.

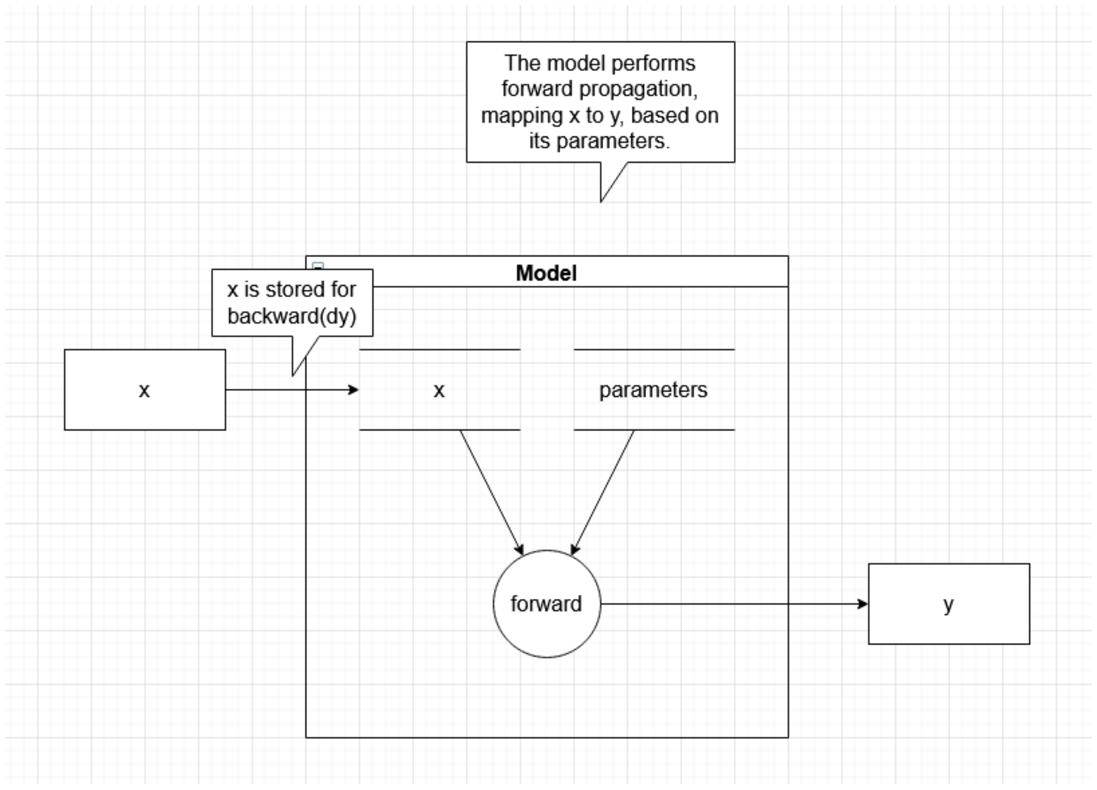
The model performs forward propagation, mapping x to y, based on its parameters.
x is stored for backward(dy)
Model
x
x
parameters
forward
y

Figure 4: Passing data through a Model.

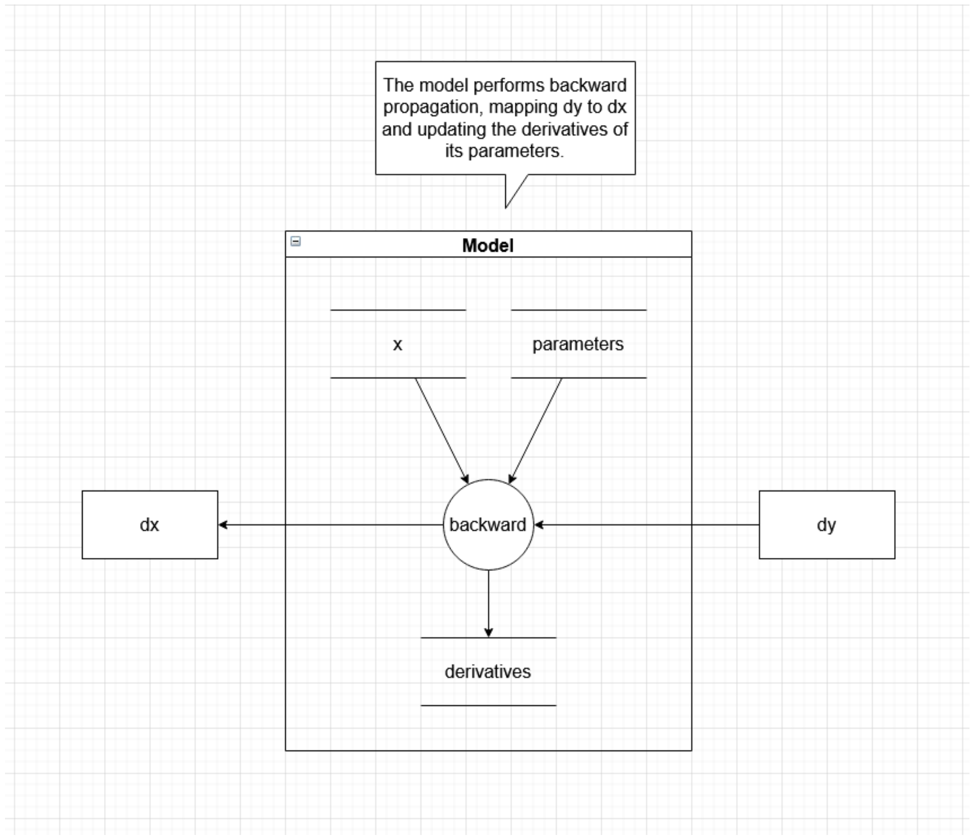
The model performs backward propagation, mapping dy to dx and updating the derivatives of its parameters.
Model
x
parameters
dx
backward
dy
derivatives

Figure 5: Updating a Model’s derivatives and backpropagating the derivatives.

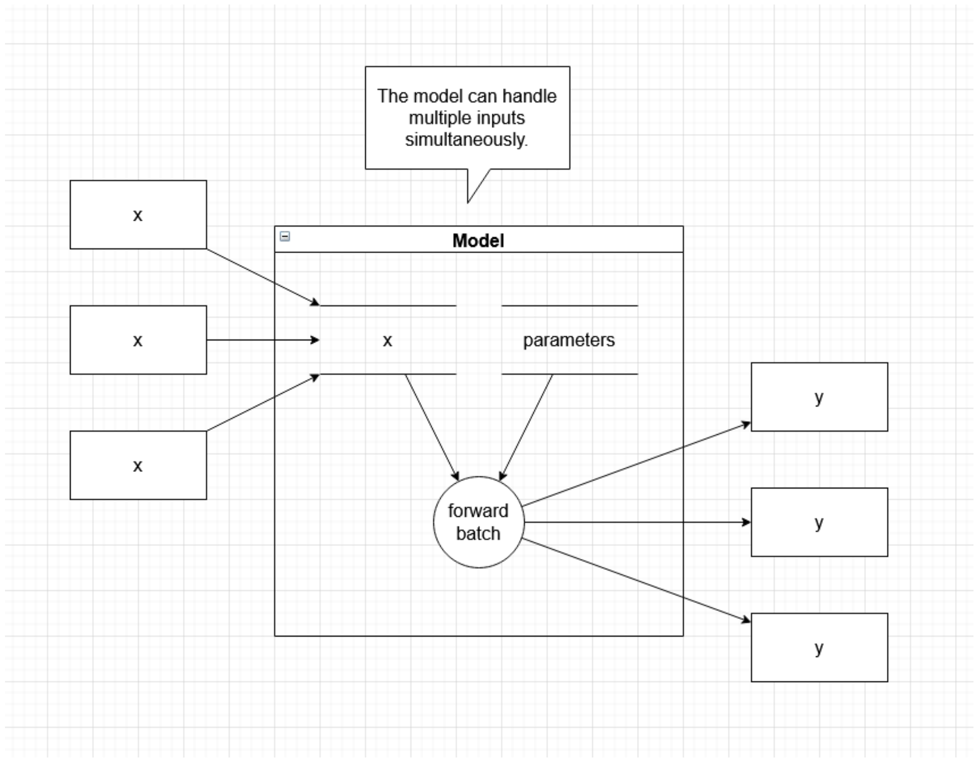

Figure 6: Intra-parallel forward propagation.

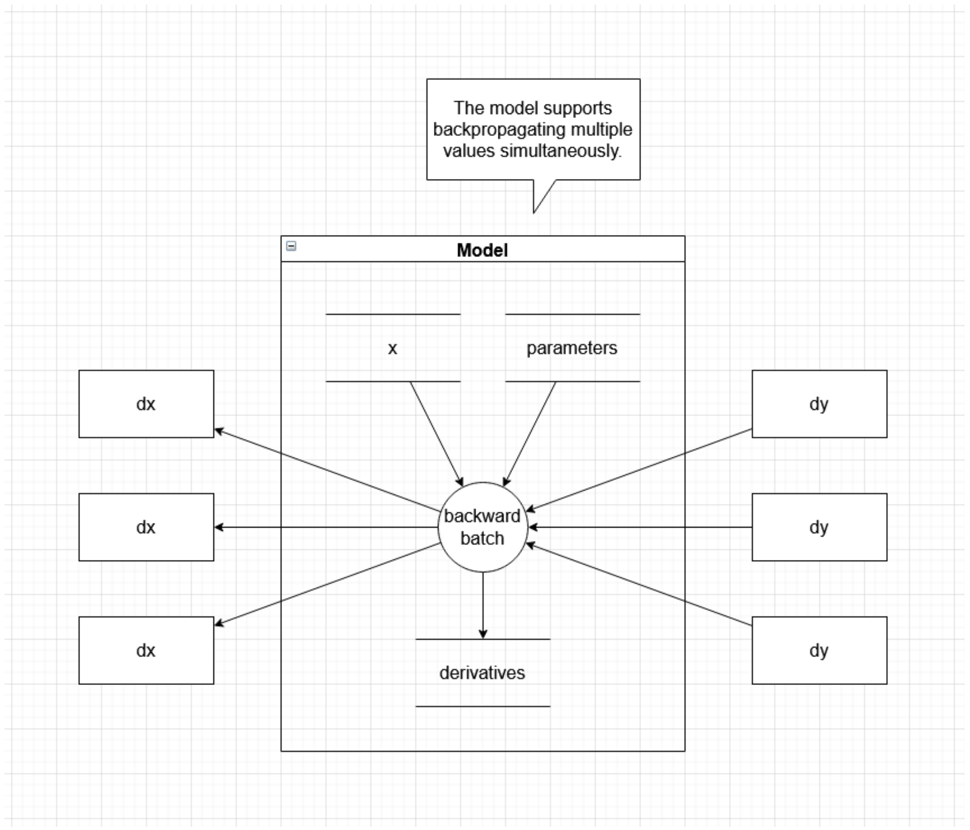
The model supports backpropagating multiple values simultaneously.
Model
x
parameters
dx
dx
dx
backward batch
dy
dy
dy
derivatives

Figure 7: Intra-parallel backward propagation.

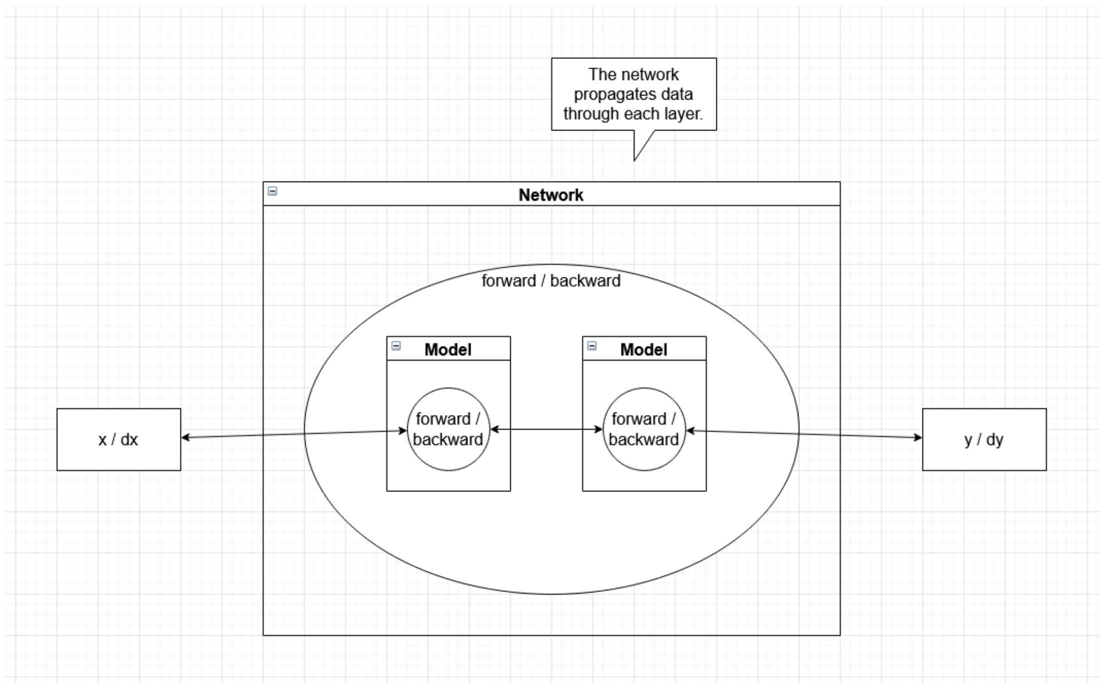

Figure 8: The flow of data during forward/backward propagation through a network by passing through multiple layers.

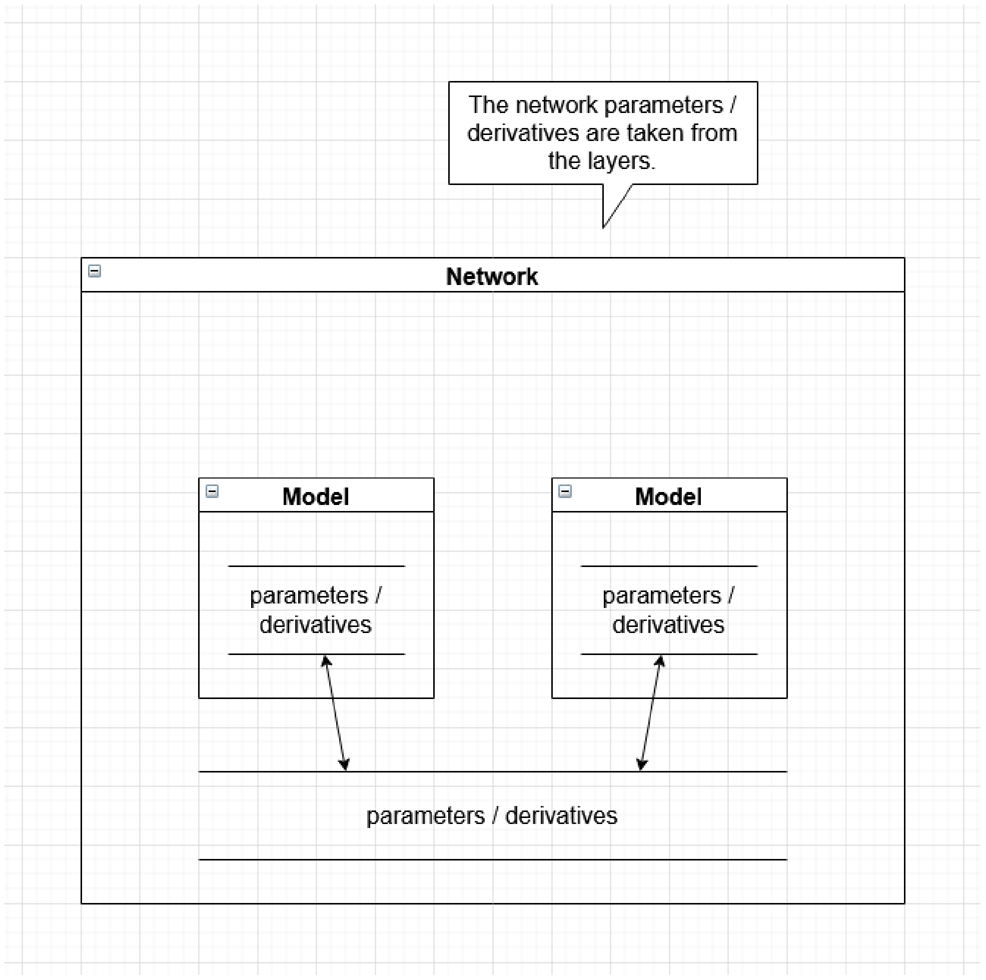

Figure 9: Networks and their layers should share the same parameters/derivatives.

Figures 8 and 9 show how whole networks interact with their underlying layers. During forward/backward propagation, data should flow through each layer in a sequential fashion. The underlying parameters and derivatives of each layer should also be shared by the overarching network. This allows the network to function itself as a model which essentially groups several "hidden" models.

5 EXTERNAL INTERFACES

5.1 Interface Architecture

No such interface exists for EasyNN.

5.2 Interface Detailed Design

No such interface exists for EasyNN.

6 SYSTEM INTEGRITY CONTROLS

EasyNN is an installable package, meaning multiple users are going to be installing and manipulating some of the usages of the different components. There is no major/critical data on any portion of the package that can be stolen.