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

TABLE OF CONTENT

INTRODUCTION **3**
- Purpose and Scope 3
- Project Executive Summary 3
  - System Overview 3
  - Design Constraints 4
  - Future Contingencies 4
- Document Organization 4
- Project References 4
- Glossary 4

SYSTEM ARCHITECTURE **4**
- System Hardware Architecture 5
- System Software Architecture 5
- Internal Communications Architecture 6

HUMAN-MACHINE INTERFACE **6**
- Inputs 6
- Outputs 7

DETAILED DESIGN **7**
- Hardware Detailed Design 7
- Internal Communications Detailed Design 9

EXTERNAL INTERFACES **10**
- Interface Architecture 11
- Interface Detailed Design 11

SYSTEM INTEGRITY CONTROLS **11**

SYSTEM DESIGN DOCUMENT

Overview
The System Design Document describes the system requirements, operating environment, system and subsystem architecture, files and database design, input formats, output layouts, human-machine interfaces, detailed design, processing logic, and external interfaces.

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

This section describes the system in narrative form using non-technical terms. It should provide a high-level system architecture diagram showing a subsystem break out of the system, if applicable. The high-level system architecture or subsystem diagrams should, if applicable, show interfaces to external systems. Supply a high-level context diagram for the system and subsystems, if applicable. Refer to the requirements traceability matrix (RTM) in the Functional Requirements Document (FRD), to identify the allocation of the functional requirements into this design document.

EasyNN is a python package designed to provide an easy-to-use Neural Network. 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

This section describes any constraints in the system design (reference any trade-off analyses conducted such as resource use versus productivity, or conflicts with other systems) and includes any assumptions made by the project team in developing the system design.

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

This section describes any contingencies that might arise in the design of the system that may change the development direction. Possibilities include a lack of interface agreements with outside agencies or unstable architectures at the time this document is produced. Address any possible workarounds or alternative plans.

If EasyNN proves to be too difficult for users, then EasyNN should be made more user-friendly by improving the documentation or software design. 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

This document is designed to give the reader an idea of the system design. The following sections will provide information on what the product does, limitations, interactions, interfaces, hardware and software designs, and security.

1.4 Project References

This section provides a bibliography of key project references and deliverables that have been produced before this point.

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.

1.5 Glossary

Supply a glossary of all terms and abbreviations used in this document. If the glossary is several pages in length, it may be included as an appendix.

1. A model is a representation of another system.

2. A(n) (Artificial) Neural Network (NN or ANN) is a simplified computational model of the human brain, loosely based on the idea of passing information between several neurons to get an output.

3. A Neuron is a component in a Neural Network that collects, stores, and sends values.

4. A loss function is a numerical measurement of the performance of a model.

5. Optimization is the process of either minimizing or maximizing a loss function.

6. Machine learning is the process of optimizing a model.

7. Forward Propagation is the process of processing values through a Neural Network, starting from the input nodes and moving to the output nodes.

8. Back Propagation is the process of computing components of the gradient iteratively instead of entirely at once, starting from the output nodes and moving to the input nodes.

9. A tensor is a 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

In this section, describe the system and/or subsystem(s) architecture for the project. References to external entities should be minimal, as they will be described in detail in Section 6, External Interfaces.

2.1 System Hardware Architecture

In this section, describe the overall system hardware and organization. Include a list of hardware components (with a brief description of each item) and diagrams showing the connectivity between the components. If appropriate, use subsections to address each subsystem.

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.

2.2 System Software Architecture

In this section, describe the overall system software and organization. Include a list of software modules (this could include functions, subroutines, or classes), computer languages, and programming computer-aided software engineering tools (with a brief description of the function of each item). Use structured organization diagrams/object-oriented diagrams that show the various segmentation levels down to the lowest level. All features on the diagrams should have reference numbers and names. Include a narrative that expands on and enhances the understanding of the functional breakdown. If appropriate, use subsections to address each module.

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.

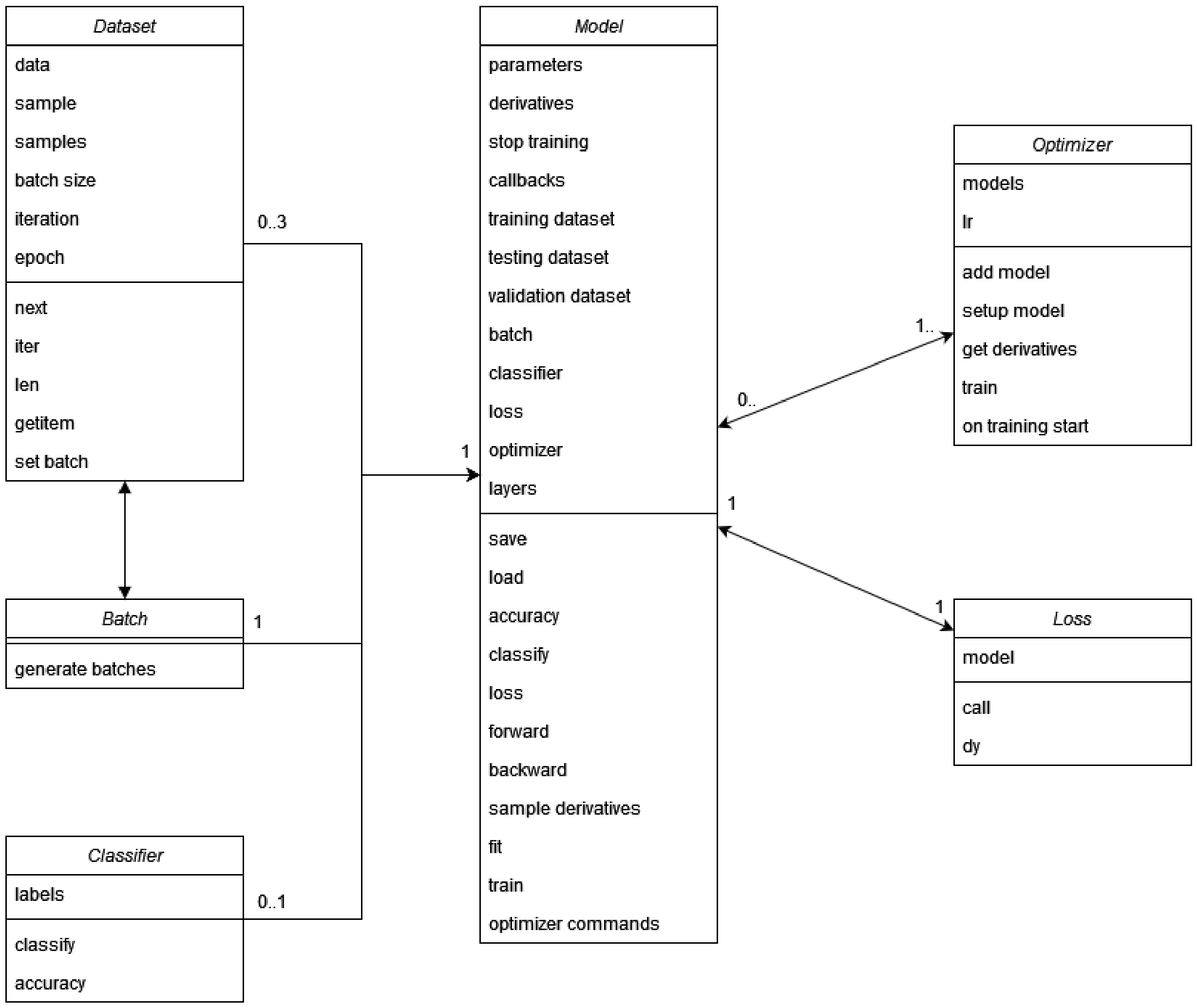


Figure 2: The class diagram.

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

This section provides the detailed design of the system and subsystem inputs and outputs relative to the user/operator. Any additional information may be added to this section and can be organized according to whatever structure best presents the operator input and output designs. Depending on the particular nature of the project, it may be appropriate to repeat these sections at both the subsystem and design module levels. Additional information may be added to the subsections if the suggested lists are inadequate to describe the project inputs and outputs.

3.1 Inputs

This section is a description of the input media used by the operator for providing information to the system; show mapping to the high-level data flows described in Section 1.2.1, System Overview. For example, data entry screens, optical character readers, bar scanners, etc. If appropriate, the input record types, file structures, and database structures provided in Section 3, File and Database Design, may be referenced. Include data element definitions, or refer to the data dictionary.

Provide the layout of all input data screens or graphical user interfaces (GUTs) (for example, windows). Provide a graphic representation of each interface. Define all data elements associated with each screen or GUI, or reference the data dictionary.

This section should contain edit criteria for the data elements, including specific values, range of values, mandatory/optional, alphanumeric values, and length. Also, address data entry controls to prevent edit bypassing.

Discuss the miscellaneous messages associated with operator inputs, including the following:

- Copies of the form(s) if the input data are keyed or scanned for data entry from printed forms
- Description of any access restrictions or security considerations
- Each transaction name, code, and definition, if the system is a transaction-based processing system

  The user will download the EasyNN package and open the run.py file. In that file they will be able to edit what they need to 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.

In order to customize features, the classes and methods provide a variety of attributes/parameters which can be interfaced according to their documentation. Custom classes may also be implemented.

3.2 Outputs

This section describes the system output design relative to the user/operator; shows mapping to the high-level data flows described in Section 1.2.1. System outputs include reports, data display screens, and GUIs, query results, etc. The output files are described in Section 3 and may be referenced in this section. The following should be provided, if appropriate:

- Identification of codes and names for reports and data display screens
- Description of report and screen contents (provide a graphic representation of each layout and define all data elements associated with the layout or reference the data dictionary)
- Description of the purpose of the output, including identification of the primary users
- Report distribution requirements, if any (include frequency for periodic reports)
- Description of any access restrictions or security considerations

The only output for the system currently are images and predictions. A sample of the output is on the GitHub wiki section and the readme.

4 DETAILED DESIGN

This section provides the information needed for a system development team to build and integrate the hardware components, code and integrate the software modules, and interconnect the hardware and software segments into a functional product. Additionally, this section addresses the detailed procedures for combining separate COTS packages into a single system. Every detailed requirement should map back to the FRD, and the mapping should be presented in an update to the RTM and include the RTM as an appendix to this design document.

4.1 Hardware Detailed Design

EasyNN has no hardware components required for it since it is a python package that is downloaded directly from GitHub.

Software Detailed Design

A software module is the lowest level of design granularity in the system. Depending on the software development approach, there may be one or more modules per system. This section should provide enough detailed information about logic and data necessary to completely write source code for all modules in the system (and/or integrate COTS software programs).

If there are many modules or if the module documentation is extensive, place it in an appendix or reference a separate document. Add additional diagrams and information, if necessary, to describe each module, its functionality, and its hierarchy. Industry-standard module specification practices should be followed. Include the following information in the detailed module designs:

- A narrative description of each module, its function(s), the conditions under which it is used (called or scheduled for execution), the overall processing, logic, interfaces to other modules, interfaces to external systems, security requirements, etc.; explain any algorithms used by the module in detail
- For COTS packages, specify any call routines or bridging programs to integrate the package with the system and/or other COTS packages (for example, Dynamic Link Libraries)
- Data elements, record structures, and file structures associated with module input and output
- Graphical representation of the module processing, logic, flow of control, and algorithms, using an accepted diagramming approach (for example, structure charts, action diagrams, flowcharts, etc.)
- Data entry and data output graphics; define or reference associated data elements; if the project is large and complex or if the detailed module designs will be incorporated into a separate document, then it may be appropriate to repeat the screen information in this section
- Report layout

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.

Add file system image here.

4.2 Internal Communications Detailed Design

If the system includes more than one component there may be a requirement for internal communications to exchange information, provide commands, or support input/output functions. This section should provide enough detailed information about the communication requirements to correctly build and/or procure the communications components for the system. Include the following information in the detailed designs (as appropriate):

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 1 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.

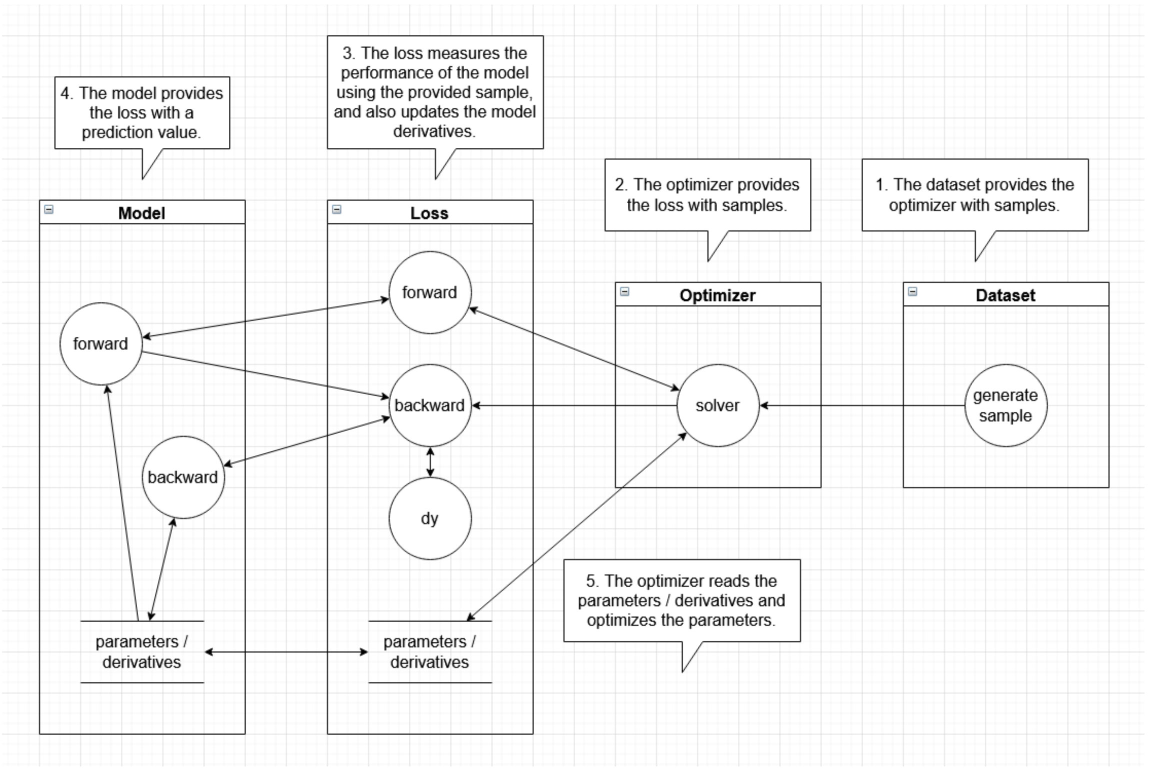

3. The loss measures the performance of the model using the provided sample, and also updates the model derivatives.
4. The model provides the loss with a prediction value.
2. The optimizer provides the loss with samples.
1. The dataset provides the optimizer with samples.
Model
Loss
Optimizer
Dataset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

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. 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 should take in inputs and return outputs by using their parameters. They should then be able to take derivatives and return derivatives, as well as update the derivatives of their own parameters. Furthermore, they should 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.

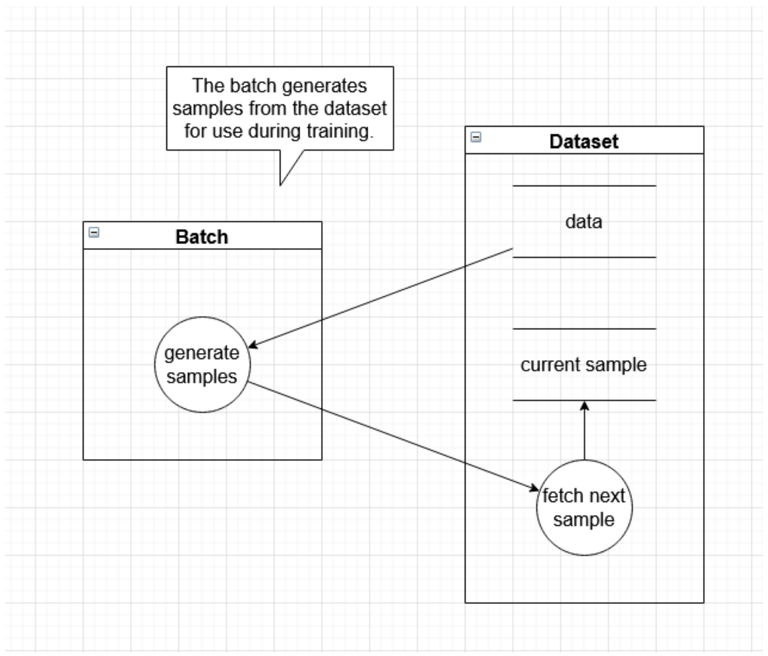

The batch generates samples from the dataset for use during training.
Dataset
data
current sample
fetch next sample
Batch
generate samples

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

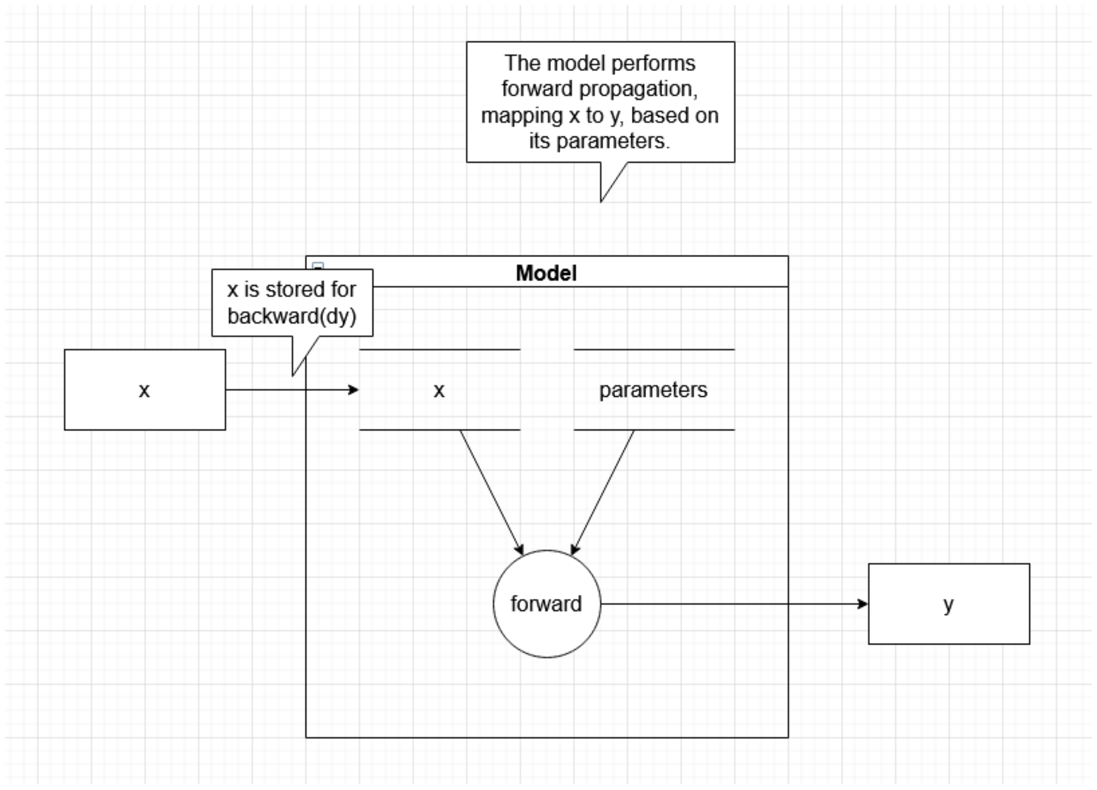
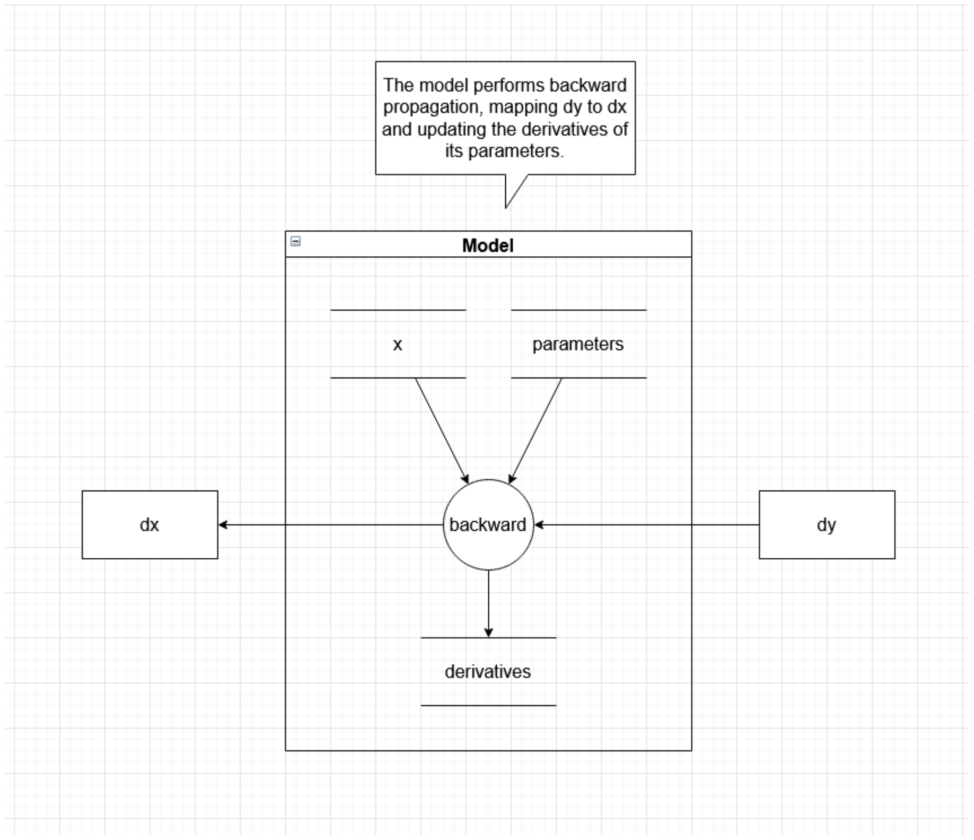

The model performs forward propagation, mapping x to y, based on its parameters.
Model
x is stored for backward(dy)
x
x
parameters
forward
y

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.

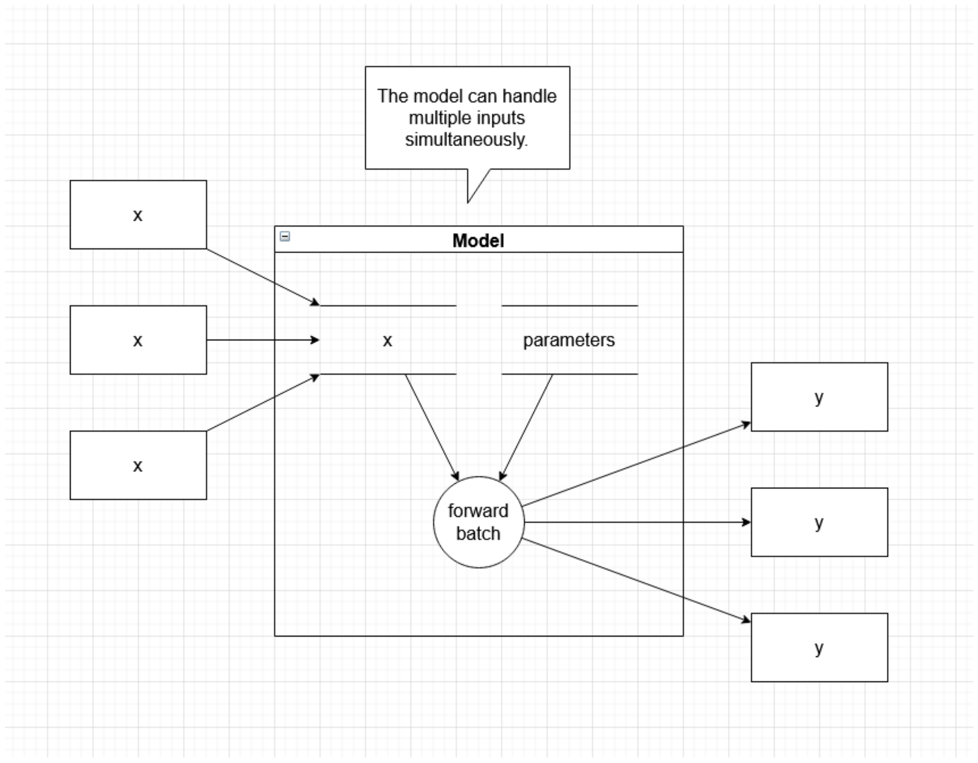


Figure 6: Intra-parallel forward propagation.

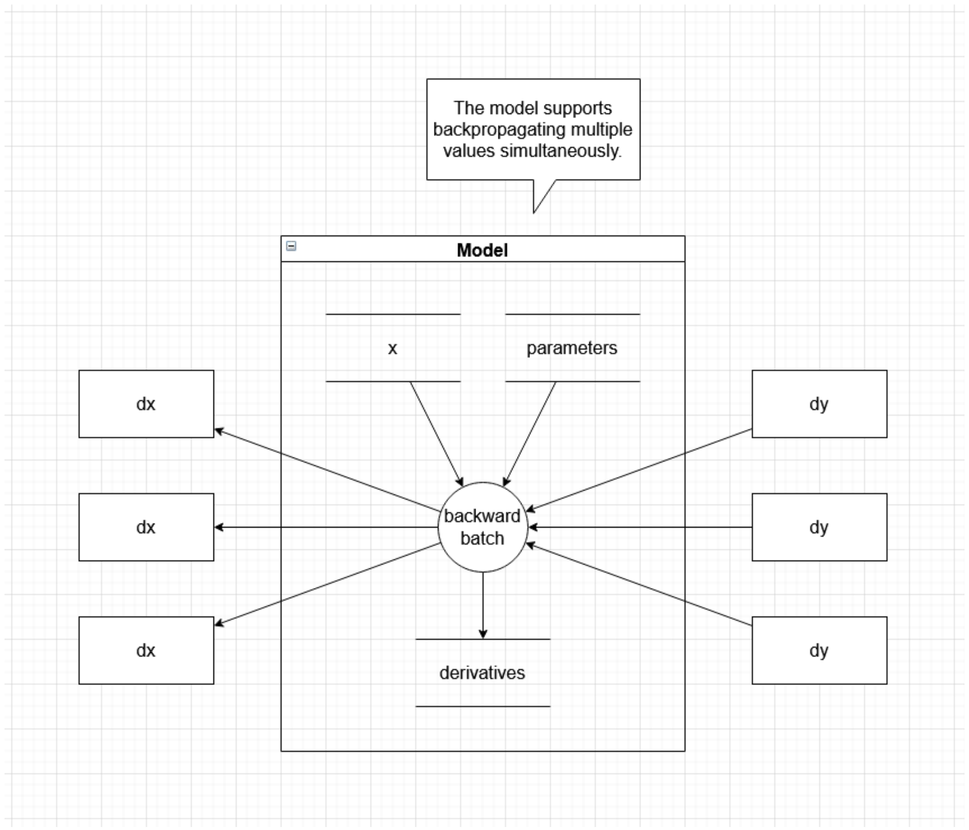

The model supports backpropagating multiple values simultaneously.
Model
x
parameters
dx
dx
dx
backward batch
dy
dy
dy
derivatives

Figure 7: Intra-parallel backward propagation.

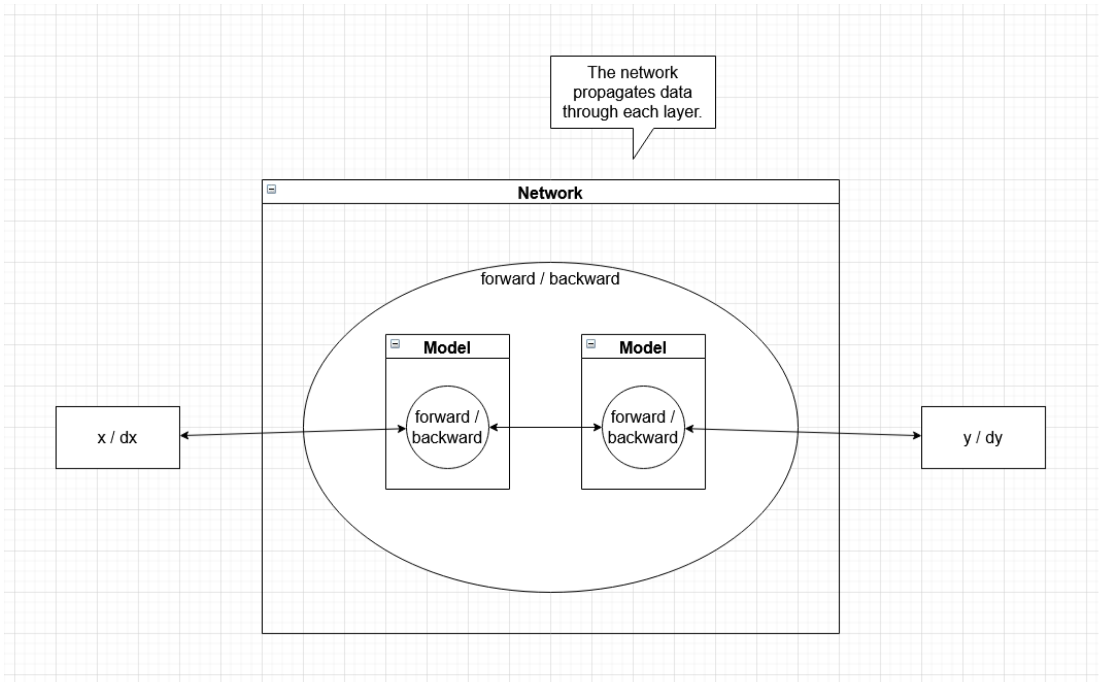


Figure 8: The flow of data during forward/backward propagation through a network by passing through multiple layers.

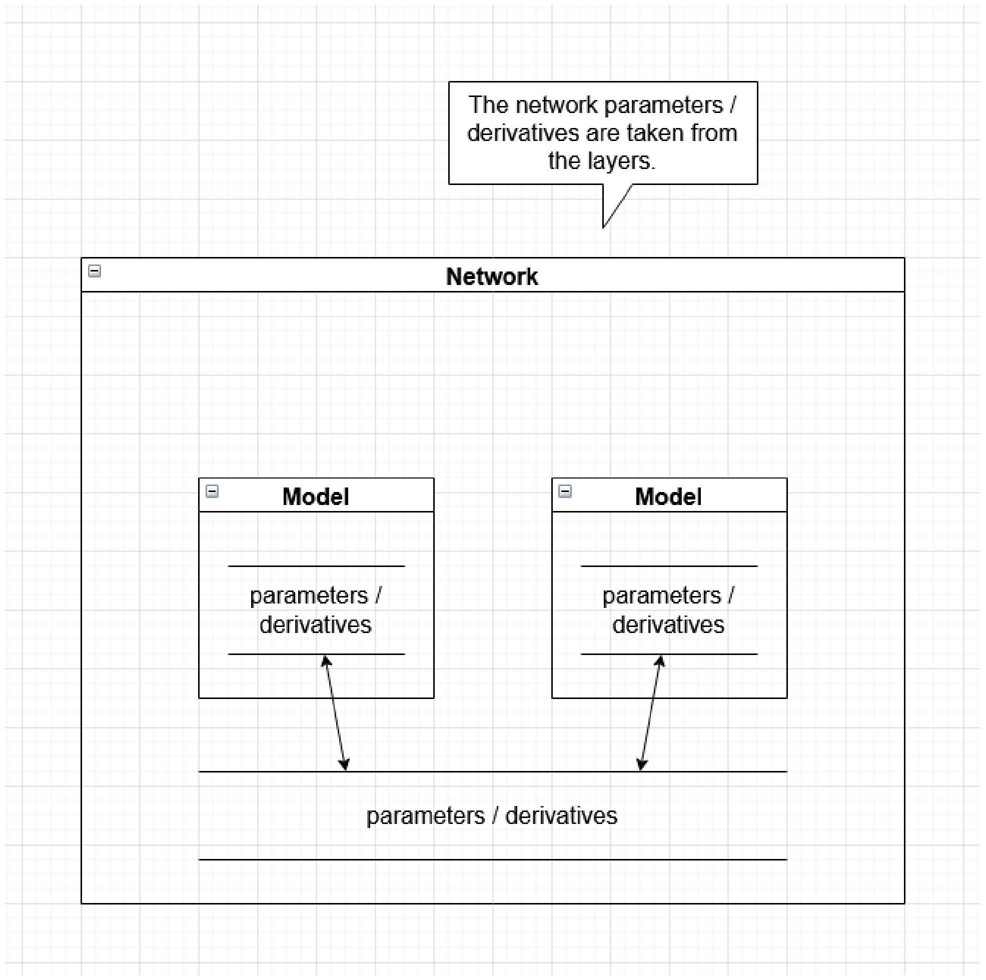


Figure 9: Networks and their layers should share the same parameters/derivatives.

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

External systems are any systems that are not within the scope of the system under development, regardless of whether the other systems are managed by the State or another agency. In this section, describe the electronic interface(s) between this system and each of the other systems and/or subsystem(s), emphasizing the point of view of the system being developed.

5.1 Interface Architecture

In this section, describe the interface(s) between the system being developed and other systems; for example, batch transfers, queries, etc. Include the interface architecture(s) being implemented, such as wide area networks, gateways, etc. Provide a diagram depicting the communications path(s) between this system and each of the other systems, which should map to the context diagrams in Section 1.2.1. If appropriate, use subsections to address each interface being implemented.

No such interface exists for EasyNN.

5.2 Interface Detailed Design

For each system that provides information exchange with the system under development, there is a requirement for rules governing the interface. This section should provide enough detailed information about the interface requirements to correctly format, transmit, and/or receive data across the interface. Include the following information in the detailed design for each interface (as appropriate):

- The data format requirements; if there is a need to reformat data before they are transmitted or after incoming data is received, tools and/or methods for the reformat process should be defined
- Specifications for hand-shaking protocols between the two systems; include the content and format of the information to be included in the handshake messages, the timing for exchanging these messages, and the steps to be taken when errors are identified
- Format(s) for error reports exchanged between the systems; should address the disposition of error reports; for example, retained in a file, sent to a printer, flag/alarm sent to the operator, etc.
- Graphical representation of the connectivity between systems, showing the direction of data flow
- Query and response descriptions

If a formal Interface Control Document (ICD) exists for a given interface, the information can be copied, or the ICD can be referenced in this section.

No such interface exists for EasyNN.

6 SYSTEM INTEGRITY CONTROLS

Sensitive systems use the information for which the loss, misuse, modification of, or unauthorized access to that information could affect the conduct of State programs or the privacy to which individuals are entitled.

Developers of sensitive State systems are required to develop specifications for the following minimum levels of control:

- Internal security to restrict access of critical data items to only those access types required by users
- Audit procedures to meet the control, reporting, and retention period requirements for operational and management reports
- Application audit trails to dynamically audit retrieval access to designated critical data
- Standard Tables to be used or requested for validating data fields
- Verification processes for additions, deletions, or updates of critical data

Ability to identify all audit information by user identification, network terminal identification, date, time, and data accessed or changed.

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