Milestone 1: Design

Kyra Patton (kp483), Agrippa Kellum (ask252)
Luca Leeser (ll698), Stephanie Sinwell (sas575)

1. System Description

We will create a program that recognizes handwritten digits drawn in a GUI, that is then classified in the backend by a neural network that we build using matrix operation optimization libraries that are available in OCaml. The program will include a simple library, allowing users to generate classifiers by inputting labeled datasets and a loss function.

Key features:

1. A basic library to allow the construction of simple fully connected neural networks
2. Usage of Lacaml (https://github.com/mmottl/lacaml) BLAS binding to optimize performance
3. Simple interfaces and clear documentation to support client usage
4. Implementation of basic loss functions and backpropagation
5. GUI that allows the user to draw digits; the saved image is transformed into a matrix as the neural network's input

We will need to find or make an implementation of matrices. Then, we can implement tensors as functions which input and output matrices. A neural net is simply a set of nested matrix operations (tensors) with a nonlinear function applied to their output. We will allow users to specify the nesting using a directed graph structure with tensors as nodes.

In order to make the neural net trainable, nodes in the network can have parameters. We will implement an algorithm for mutating these parameters based on the gradient of the loss function across the parameter space given a learning speed. Iterative mutation will produce parameters that minimize the neural network's loss for the given inputs. To allow for fast matrix computations, we will be using the previously mentioned OCaml matrix optimization library.

There already exists a library that implements ANNs in OCaml, called Owl. The documentation is not clear about how full the implementation is, but we do not intend to look at this package while developing our project.

2. System Design

a. Layer

A neural network is made up of layers, where a layer is a matrix with an activation function that can be applied to it. This module contains types to represent layers (implemented as matrices) and activation functions. It also contains functions that can be performed on layers and matrices, such as matrix multiplication or the application of an individual layer's activation function. This module forms the basis of the neural network as a whole, which relies on matrix multiplications to function.

b. Loss

This modules contains a data type representing loss functions. The different kinds of loss functions we have included are described in greater detail in Part 4 (Data) below. The process of training a neural network consists of minimizing a loss function through methods like gradient descent.

c. Model

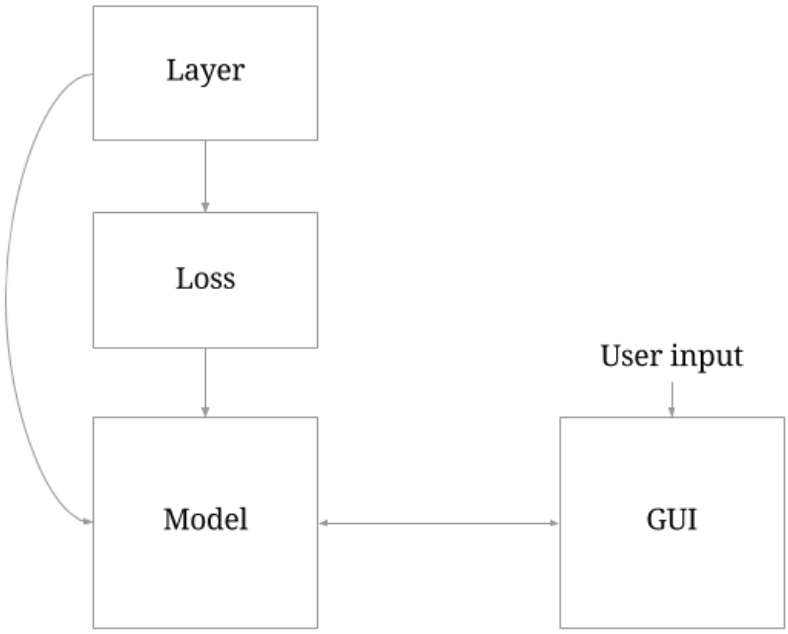
This module contains the model type, which organizes layers in a sequence. It will contain a function for propagating signals through the layers, ultimately producing categorizations. It will also contain the function for backpropagation-- the process through which layers update themselves towards producing more predictive outputs.

d. GUI

The main purpose of the GUI is to allow for users to draw handwritten digits, which will then be fed into the neural network and classified. In this module, the user-drawn image is converted to a matrix and fed into the neural network in the Model module. The Model module then classifies the user-drawn image as a digit; this output is returned to the GUI module, which will display the output of the classification process to the user in a text field.

The GUI module contains all the functions necessary to support the interactive drawing application. The GUI itself will be a window, containing a drawing pane, a classify button, a reset button, and a text field that displays the output of the neural network after classification. The drawing pane will have a black background, and users will be able to draw with a fixed-width white drawing tool, to mimic the appearance of the MNIST images. MNIST images are 28x28, but we plan to have the drawing panel be larger and later scaled down to 28x28 at classification time to make the interface more user-friendly.

A block diagram highlighting how the modules connect can be seen below.



3. Module Design

See our .mli files for more details about each module.

4. Data

a. Matrix

A matrix data structure will be used to represent our neural network model. We have defined a matrix to be a list of float lists, where every entry should be a float between 0 and 1.

b. Tensor

We defined an additional data type for tensors, which are three-dimensional matrices (i.e. lists of lists of float lists). However, as of writing the design document, we are not clear on whether we will be using matrices or tensors to implement everything.

c. Loss

The loss data type contains constructors for different types of loss functions. The loss function can be L1, which is the Manhattan distance between two vectors; L2, which is the Euclidean distance between two vectors; and categorical cross entropy, which is the difference in probability distributions of the two vectors. Each element in these vectors is a probability distribution of a certain category.

d. Activation

The activation data type contains constructors for different non-linear activation functions. An activation function can either be sigmoid, ReLU, or softmax.

e. Layer

The layer data type represents a layer in the neural network. A layer contains a matrix and an activation function. It is possible that we will develop multiple kinds of layers, such as Convolutional layers.

f. Model

A neural network consists of several layers. Our model datatype represents an entire neural network, and is therefore a list of layers.

g. External datasets

We will use the MNIST dataset of handwritten numbers for training and testing our neural networks. MNIST is available in OCaml through this Github repository: https://github.com/rleonid/dsfo

5. External Dependencies

a. Lacaml

Lacaml (Linear Algebra for OCaml) is a BLAS binding for OCaml which we will use to implement the matrix functions. High performance linear algebra software such as BLAS is many orders of magnitude faster than natively implemented functions for linear algebra; a

native solution would not be viable for a regular sized neural net. All relevant usage of Lacaml will be exposed in the Layer module.

b. Lablgtk

For the GUI, we will be using parts of the OCaml Graphics module, but we will also utilize Lablgtk. Lablgtk is the OCaml interface to GTK+, a toolkit for making graphical applications. Lablgtk will be used for the GUI window and the graphical components within the window, such as the drawing canvas, as well as buttons that will allow users to classify their drawn image or reset the canvas. The specific modules we will be using are GMain and GMisc, which contain the graphical components that we would need for the GUI.

6. Testing Plan

In the initial implementation phase of the neural network, we will write unit tests to ensure that each of the modules is working separately. For instance, in the Layer module, we will require black box tests to ensure that multiplying matrices and performing element-wise activations are working properly. In the Loss module, black box tests will also be written to ensure that performing a loss function on the given matrices produces the correct output.

When basic matrix operations and the loss functions are functional, we will implement non-linear layers and back-propagation on very small networks to ensure that our algorithm works properly. This could be verified once more through black box testing, based on the specifications in our interface. As we approach the final phases of implementation, we will be able to build more complex neural networks. Ultimately, our goal is to train a neural network on a training set, which will be a subset of the MNIST dataset. To verify correctness of the network as a whole, we will input a testing set, which is a different subset of the MNIST dataset. If the neural network is able to classify the images in the testing set correctly, then we will know that the neural network is functional and that overfitting has not occurred.

We will test the GUI interactively while implementing its functionality. After the GUI is working, we plan to have someone outside of the class use it to ensure that the interface is clear and usable. We can then use their feedback to improve the user experience of the GUI.

To make sure that every member of the team is accountable for testing and writing correct code, we will make sure that everyone contributes to writing part of the test suites. Additionally, we will ensure that everyone writes good specifications of any additional functions they write, and that unit tests on these helper functions will be added.