SDD document for Stochastic Optimization & Visualisation

April 12, 2021

1 Introduction

1.1 Purpose

This software design document describes the architecture and system design of a software aimed for visualising the stochastic gradient descent algorithm for finding the optimal minima of a given function in a certain range under different optimizers. The graphic visualisations provided by the software can prove beneficial for understanding how the algorithm works under different optimizers and other settings.

1.2 Scope

This is a software which will help in the visualisation of effectiveness of various optimizers under different hyperparameters, loss functions etc. Any user can use this software to perform stochastic gradient descent algorithm on some pre-chosen loss functions and visualise the algorithm finding the optimal point step by step which minimises the loss function.

This software is assumed to be primarily used by students (both college and school going) and teachers (schools and colleges). For students, it can be used as a means of understanding the working of optimizers well, while for teachers it can serve as a way of demonstrating the concepts while teaching. We assume the product will be most helpful for people who already have the knowledge of how gradient descent works in order to interpret the results correctly.

1.3 Overview

The document is organised in the following manner ahead

- **Section 1.4** - Discusses about the concepts and definitions which will be required in order to interpret the design provided ahead.

- **Section 2** - Gives a general description of the functionality, context and design of our project. Provides any background information if necessary.
- **Section 3.1** - This describes the architectural design. Provides a high level overview of how responsibilities of the system were partitioned and then assigned to subsystems.
- **Section 3.2** - Provides a decomposition of the subsystems in the architectural design.
- **Section 3.3** - Discuss the rationale for selecting the architecture described in 3.1 including critical issues and trade/offs that were considered.
- **Section 4** - Describes how the major data or system entities are stored, processed and organized.
- **Section 5** - In this section, we take a closer look at what each component does in a more systematic way. Provides pseudo codes for some of the algorithms used in the functions descrives in 3.2
- **Section 6.1** - This section explains how the user will be able to use our system to complete all the expected features and the feedback information that will be displayed for the user.
- **Section 6.2** - Displays screenshots showing the interface from the user's perspective.
- **Section 6.3** - A discussion of screen objects and actions associated with those objects.
- **Section 7** - Provides a cross reference that traces components and data structures to the requirements in your SRS document.
- **Section 8** - This section gives the timeline to be followed during the project.
- **Section 9** - This is the appendix section containing any further details which are not much relevant to understanding the design document but can be looked at for further details.

1.4 Definitions

Some of the definitions are provided below.

- **Optimization:** Mathematical optimization (alternatively spelled optimisation) or mathematical programming is the selection of a best element, with regard to some criterion, from some set of available alternatives.
- **Gradient Descent:** Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea is to take repeated steps in the opposite direction of the gradient (or approximate gradient) of the function at the current point, because this is the direction of steepest descent.

- **Stochastic Gradient Descent:** Stochastic gradient descent (often abbreviated SGD) is an iterative method for optimizing an objective function with suitable smoothness properties (Differentiable etc.). It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data). This is extremely useful in topics like Machine Learning, where we need to find minima of a loss function without having to do too much computation.

The mathematical algorithm for Stochastic Gradient Descent is provided in the **Appendix** section in the document.

2 System Overview

The system to be designed will primarily be for a software that opens up a webpage on the user's default browser and the server will be hosted on the user's machine which deploys the user's hardware resources for computing the functions in the backend.

The user will provide his choice of parameters using the frontend of the webpage (GUI) and then will ask to visualise the stochastic gradient descent algorithm. The algorithm should run in the backend and then return a list of generated points and value of loss function at those points which will be used by the frontend to animate the SGD algorithm and display the same.

We plan on using the following libraries and frameworks for the project

- **Visualization:** matplotlib and imageio
- **Web interface:** Javascript
- **Frontend Libraries:** ReactJS
- **Backend framework/libraries:** PyTorch, numpy and ExpressJS

3 System Architecture

3.1 Architectural Design

This section will describe the various modules present in the program in an overview and the relationships among them.

The first script we will be providing to the user will be *requirements.txt* file containing the dependencies used for the project and install script for each of the dependencies. The script will check if the required dependency is present or not and downloads in case it is not present.

The first module (let's call it **setup**) will be the one that will launch the application inside the user's default browser and render the frontend and set up the backend server on user's machine. This can be easily achieved using NodeJS which provide with the functionality to setup local-host servers easily on the desired port using libraries like expressJS. Now, the next thing is to design the frontend and backend of the app.

The system design will be separate for the frontend and backend part of the software. I will be enumerating the modules present in the frontend and backend part separately and then describe the relationships among them.

- **Frontend:** This part should contain two different modules as listed below.
  - **Window -** This module will be responsible for the UI layout that the user will see when the webpage opens. It also includes all the widgets like selector for various parameters, text box for taking input commands etc.
  - **Plot_area -** This is the module that will trigger when backend sends back the list of points taken over various iterations and loss functions at those points as described in the SRS doc and it will prepare the GIF using imageio and matplotlib libraries in python.
- **Backend:** Backend of our software besides the local-host server set up using NodeJS will consist of a single module (let's call it **Gradient_descent**) which will be responsible for running the gradient descent algorithm with the user provided hyperparameters and chosen optimizer. We will be using frameworks like PyTorch-cpu and librarires like numpy to make the program faster.

SPECIAL NOTE: In addition to all this, the code should strictly follow a certain style guide which will be Black for python and prettier for JavaScript.

Now, we will describe the relationships among those modules and how the data will flow through in sequence below.

- The user first starts the application using the **setup** module and starts the local server on the user provided port. This functionality can be achieved by issuing the right commands on the command line/command prompt.
- After running the application, the server and frontend will get setup, the user can then open the webpage on the browser of his choice and start visualising.
- When issuing any command or choosing parameters for the SGD algorithm, the **Windows** module will be responsible for registering all the options that the user provides.
- After the user is done providing the necessary options and submits the data, it goes to the backend.

- In the backend the **Gradient_descent** module gets the data and runs the algorithm and produces a list of generated points and corresponding loss functions.
- The list is sent to the frontend, where the **Plot_area** module comes into picture and returns the GIF which is then rendered on the user's webpage using the **Windows** module.
- All further commands like saving the GIF will be handled by the **Windows** module by sending saving requests through the server to the **Plot_area** module.

The following graph represents the data flow across various modules according to the points enumerated above.

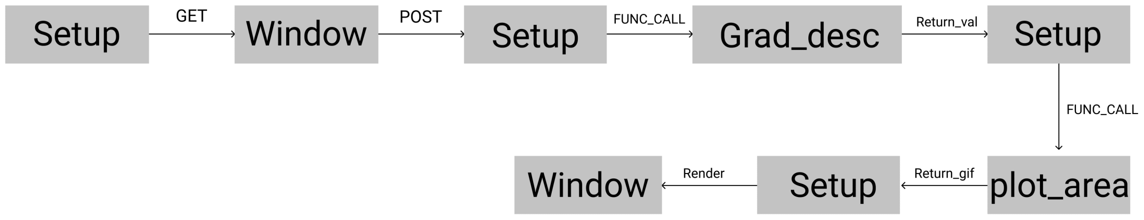


Figure 1: Data flow Diagram

3.2 Decompostion Description

Let's first discuss the **Gradient_descent** module. This module will mainly consist of the only function **Compute_SGD** which runs the gradient descent algorithm and returns the list of generated points and loss function to the frontend for rendering. The limit to the number of iterations should be kept as 200. This function contains implementations of various optimizers that we will be providing and the right one can be chosen accordingly using a switch-case or if-else statements. It also contains a function called **Check_valid** which checks for the presence of explosions and validity of the loss function's value. We should stop and throw error message if the value is not valid.

Similarly, for the module **setup** which will essentially be a server made using expressJS library. It will contain APIs to handle get requests and post requests coming from the homepage of the webapp. The initial get request coming from homepage should lead to rendering of components from **Windows** module. The post request with the user provided parameters will be handled by the running the Compute_SGD function and the values returned by the function will be sent to the **Plot_area** module. In addition to these routes, it should also have a route for saving the data where it will be accepting get requests and it should save the recently rendered GIF as well as corresponding data arrays on user's computer. This route just saves the gif in '.gif' format and the data arrays as '.txt' format and returns "200 OK" status back to the **Windows**. Besides these API request handlers there can be various utility functions which the programmer can write as per their convenience.

Now, let's discuss about the **Windows** module. This module contains essentially a react component which will work as a form through which the user submits the options he wants to submit. In addition to this form, a user will also be able to choose the parameters using a text box which can be used to enter commands and a bigger text box below this command box which will instruct user on using the command text box (for more details refer section 6.3) which explains the function of this bigger box. This will form a separate react component and may be built using simple switch cases depending on the command provided. We can have a separate function **handle_commands** which will be activated when someone writes something on command box and presses enter. We plan on implementing the following shortcuts for now.

- **Help(h):** to open the help tab (which will show them an example of how to use the site with a step by step example and will also have the shortcuts which we provide). This command can be applied by entering (h) on the command box.
- **Plot(Shift+Enter):** as soon as user enters all the data and press 'Shift + Enter' the graph plotting should start. The user need not be present on command box for this shortcut.
- **Choose Optimizer(o):** a shortcut to open the drop down menu to choose the optimiser that is to be used. This command can be applied by entering (o) on the command box. After choosing the optimizer the user will be asked to choose the hyperparameters for it. These are to be entered in the same command box, where user can type 'cust' command in order to select the custom value which we will set based on optimizer, otherwise it can be changed accordingly.
- **Choose loss function(l):** a shortcut to open the drop down menu to choose the loss function that is to be used. This command can be applied by entering (l) on the command box.
- **Save Shortcut(s):** Will save a gif of the plot(that is how with different iterations the function reaches which point) and also the data that was used to make the plots. The user need not be present on command box for this shortcut.
- **Toggling between different numeric fields(Tab):** learning rate , range to plot, starting point, function that is to be optimised. This toggling can be used once you are using the form provided above. On clicking (tab) you can toggle to different parameters.

For implementing the key presses which are not related to command box, implement a function **detect_key_press**, this should be checking on DOM if user presses certain special commands like 'Shift + Enter' and then act accordingly.

After the user submits the details, there need to be a checker function which needs to validate the input given by the user and check if all the required parameters are provided or not. If not, it should log the appropriate message on the client side. If everything goes fine, it generates a post request on the home route of our local host server to register the details and send them to the backend. In addition to all this this module should contain a restore button that restores the status of the webapp to how it was originally.

This module also contains the containers for the GIF rendering. The server when receives GIF from the **Plot_area** module renders the GIF on the webpage. After this the **windows** module also contains the code for enabling action on hovering using the following function.

- **hover_action:** This function is responsible for creating the action when cursor is hovered over a point. It first checks inside DOM if a graph is already plotted then it activates itself and produces the desired effect of showing the data relevant to that point (given in SRS) on hovering of the cursor.

Now, what's left is the last and the final module **Plot_area**. This module receives the list of points which are to be plotted, the loss function chosen, the range chosen. This module can be divided into three functions which are given below.

- **generate_figure:** This function should plot the contour of the loss function and then embed the points generated on the contour and generate frames containing the last plotted and the current point with spline. These frames are sent to the other function that will generaate the final GIF.

- **generate_GIF:** This function takes in the sequence of frames and generates an animated GIF from this. We will be using imageio library for this work. The final GIF prepared is then rendered on the DOM at appropriate place by sending it back to server which further renders it using the GIF component in **Windows** module.

The following graph summarises the system decomposition which was discussed above.

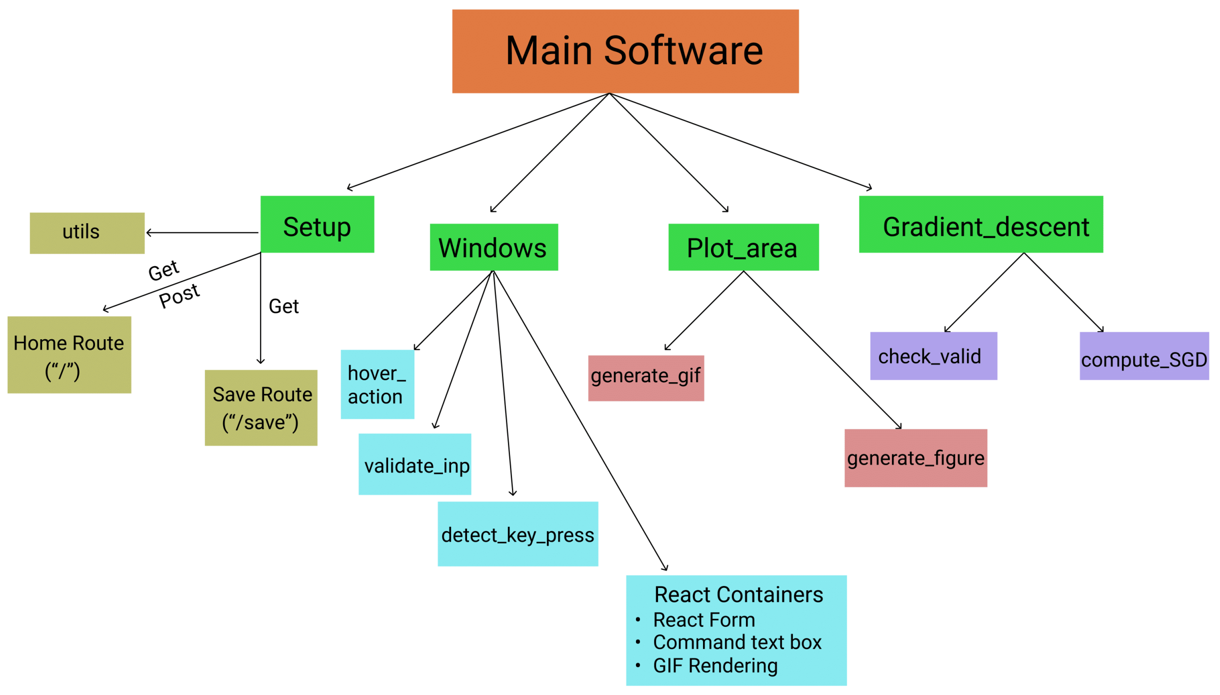


Figure 2: System Decomposition Diagram

3.3 Design Rationale

The proposed design works fine for the provided SRS and meets all the requirements. In addition, the way the modules are divided allow different teams to work parallely (most of the setup, windows and Gradient_descent modules can operate independently) and efficiently. Another feature is that most of the modules will take approximately equal time to get completed. The used libraries are all light weighted and the use of implementations in light frameworks like PyTorch-cpu allow the implementations to be blazingly fast as per the requirements. In addition all the modules divided, work in one particular language only which defines a reasonable enough rationale for the chosen design.

4 Data Design

This section describes how the data is interpreted inside the modules. The parameters provided by the user are converted in the following format

- **Function:** The user selects a string from a given list of pre-chosen functions. This is then converted to an integer for the ease of using it ahead. The integer lies in the range of [0,number_of_functions). The frontend will only accept if the provided string is from the set of provided functions, otherwise it displays an error on submission of form.

- **Optimizer:** The user selects a string from a given list of pre-chosen optimizers. This is then converted to an integer for the ease of using it ahead. The integer lies in the range of [0,number_of_optimizers). The frontend will only accept if the provided string is from the set of provided optimizerss, otherwise it displays an error on submission of form. The hyperparameters are passed as float values to the backend. Proper sanitation is provided for floats as well to check that user doesn't provides things like "NaN" or "Inf".

- **Range to plot:** This will be passed as two separate float values to the backend, one will be the start point and other will be the end point of the range. Proper sanitation is provided for floats as well to check that user doesn't provides things like "NaN" or "Inf".

- **Start point:** This is also passed as float data type to the backend. Proper sanitation is provided for floats as well to check that user doesn't provides things like "NaN" or "Inf".

- **Number of iterations:** This should be passed as an integer data type to the backend. The minimum value should be 1 and the upper limit should be 200 on the number passed. Throw an error otherwise.

The other data which moves around the modules is the list of generated points by SGD algorithm and their loss functions respectively. These will be set as numpy arrays with data type as float. The other non trivial data type was saving of frames to produce final GIF. These frames will also be stored as an array of plt objects of matplotlib. The final GIF produced will be returned as a buffer stream to the server which will then be appropriately converted into image by converting it to base64 string.

There is no need to save the functions in database and their implementations can be embedded inside the code within the switch case or if-else statements.

5 Component Design

In this section, we take a closer look at what each component does in a more systematic way. We will try to summarise some important functions present in each of the modules as explained in Section 3.2.

For **Gradient_Descent** module, we will need all the user provided data as input. Then we will decide which optimizer to use with SGD. For now, we will be providing the user with 5 optimizers. The mapping of optimizers to int will be as follows $M = \{$Adadelta :

$0, \text{Adagrad} : 1, \text{Adam} : 2, \text{Momentum} : 3, \text{RMSProp} : 4\}$. We will also store the implemetations of the various loss functions that we plan on providing in this module. The optimizers and the loss function given by the user can be used in this module by using nested switch statements (the outer one searching for the function and the inner one searching for optimizer). The next step is just to feed the hyperparameters obtained from user's inputs along with the chosen loss function and optimizer to pytorch-cpu's pre-implemented algorithm. The algorithm is to be run for number of iterations which is provided by the user and losses and the points are needed to be stored in a normal python array which will be then returned in the end.

To reduce the chances the chances of "Gradient explosion", techniques like Gradient-clipping should be used inside the function. Proper sanitisation should be done for the values of loss values if they come out to be ("Inf" or "NaN") in which case some special signal (say '-1') indicating error in calculating SGD will be returned.

For the **setup** module, we will have to implement the main server to accept the requests and render results. The get request from the home route ("/") of the webpage will be accepted by the server and in return the server will render the containers present in the Windows Module that will lead to appearing of GUI on the webpage. The DOM is rendered by the react containers now and the frontend is set up now. This is the work of the get request handler to be written using ExpressJS.

Now to handle post requests, there has to be some switch case statements here as well which will select the appropriate optimizer function and correspondingly choose the hyperparameters to be passed along to the **Gradient_descent** algorithm. This in turn returns lists of points and loss function values at those points. If it returns some special code (-1), then handle the error by logging in error message on user's screen. These are then passed to the **Plot_area** returns the produced GIF and this is then rendered on the DOM using the container for the GIF in **Windows** module. Since, the code is modular in nature we can easily run the python script using express server only by spawning a child process and running the python script with the given arguments and on returning the message do further task with this returned value.

It contains another route "/save" which will be used for saving the data and the GIF on user's computer. The GIF will be then saved on the user's computer in '.gif' format and the data arrrays will be stored in '.txt' format.

For the **Windows** module, we need to clearly define the functions of each component. The work of the GIF window component is clearly described in Section 3.2, so we will not repeat it here. The work of the shortcuts and other component is also clearly defined over there in Section 3.2. Normal components involve react forms and on submission send the post request to the local host server. The command box requires a bit attention. When the user is done writing the command and presses "Enter" key, the program should detect this key press and based on some switch-case logic implement the logic which is required further (bringing down the corresponding drop-down box or turning the command box into a text box for accepting hyperparameters for the optimizer).

These can be easily achieved by making sure that the react components of command box and the form interact with each other. The other thing that needs to be ensured by this module is the security aspect from frontend, which should ensure that user inputs the correct inputs. So, proper sanitisation needs to be ensured in the frontend as explained in Section 3.2 inside each switch-case statement and also on each entry in the react form (form is the same as described in Section 3.2). The last function of this module left to be discussed is of saving the GIF when the user enters the command 's' on the command text box. This will lead to sending a request to the server on the "/save" route. The server contains the currently rendered GIF and data arrays. It needs to save the gif in '.gif' format and data arrays as '.txt' format and return "200 OK" status.

The last module **Plot_area** is explained with quite details in Section 3.2. Refer to it for details on this module.

6 Human Interface Design

6.1 Overview of User Interface

This section describes the functionality of the system from the user's perspective. This section acts as kind of how user will be able to use the system to complete all the expected features and how the information will be rendered on his end.

The user can run the binary file for our application from the command line/command prompt and specify the port he wants to setup the server on. Then after running the program he should get a success message on the console and the message should display the link where the server has been hosted.

Next, the user opens his preferred browser and in return sees the webapp. On the window the user should see a form containing various fields with a drop-down list to select from. The form should contain these elements in order:

- **Loss function:** Contains a list of pre-chosen loss functions.
- **Optimizer:** Contains a list of pre-chosen optimizers.
- **Hyperparameters:** This section will be subdivided into various subsections. Each subsection refers to choosing hyperparameters for one of the optimizers. To see this detail, refer Sec 6.2 on screen images.

- **Left end of range:** This refers to the lower end of the range we want to visualise SGD on.
- **Right end of range:** This refers to the upper end of the range we want to visualise SGD on.
- **Starting point:** This refers to the point to start SGD from.

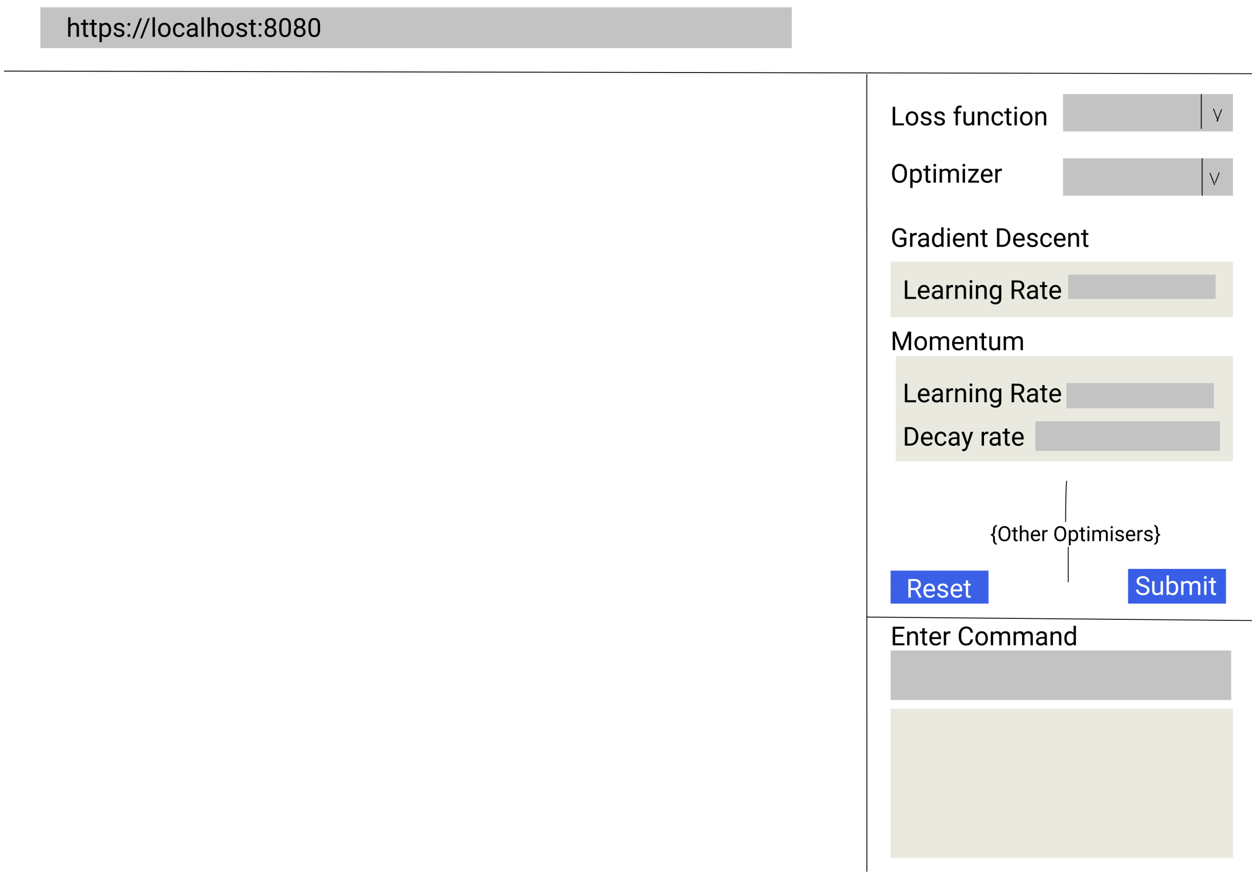
To perform similar functions using keyboard, we have a command box as well, it's specifications are pretty well explained in Section 3.2.

After doing all this, the user can either press 'Submit' button or press 'Shift + Enter' to run the program in the backend and generate the results of the optimisation. You will receive a message "Preparing Visualisation" after submitting. After the work is done, message is removed and GIF is rendered on the screen.

The user will also be provided with a 'Reset' button to reset the fields to their 'None' values.

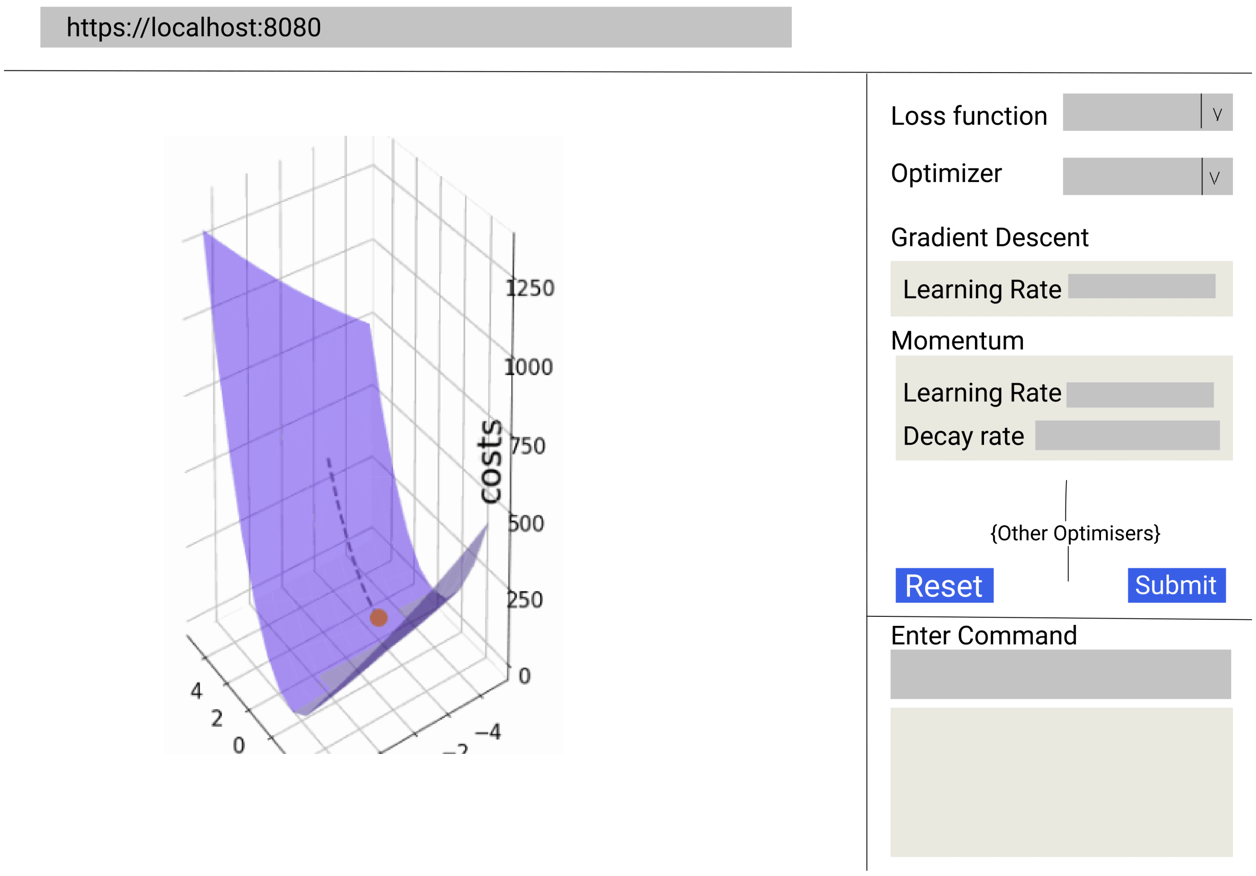
6.2 Screen Images

This section contains screenshots showing the webapp interface from user's interface. This represents the initial layout that the user will see.

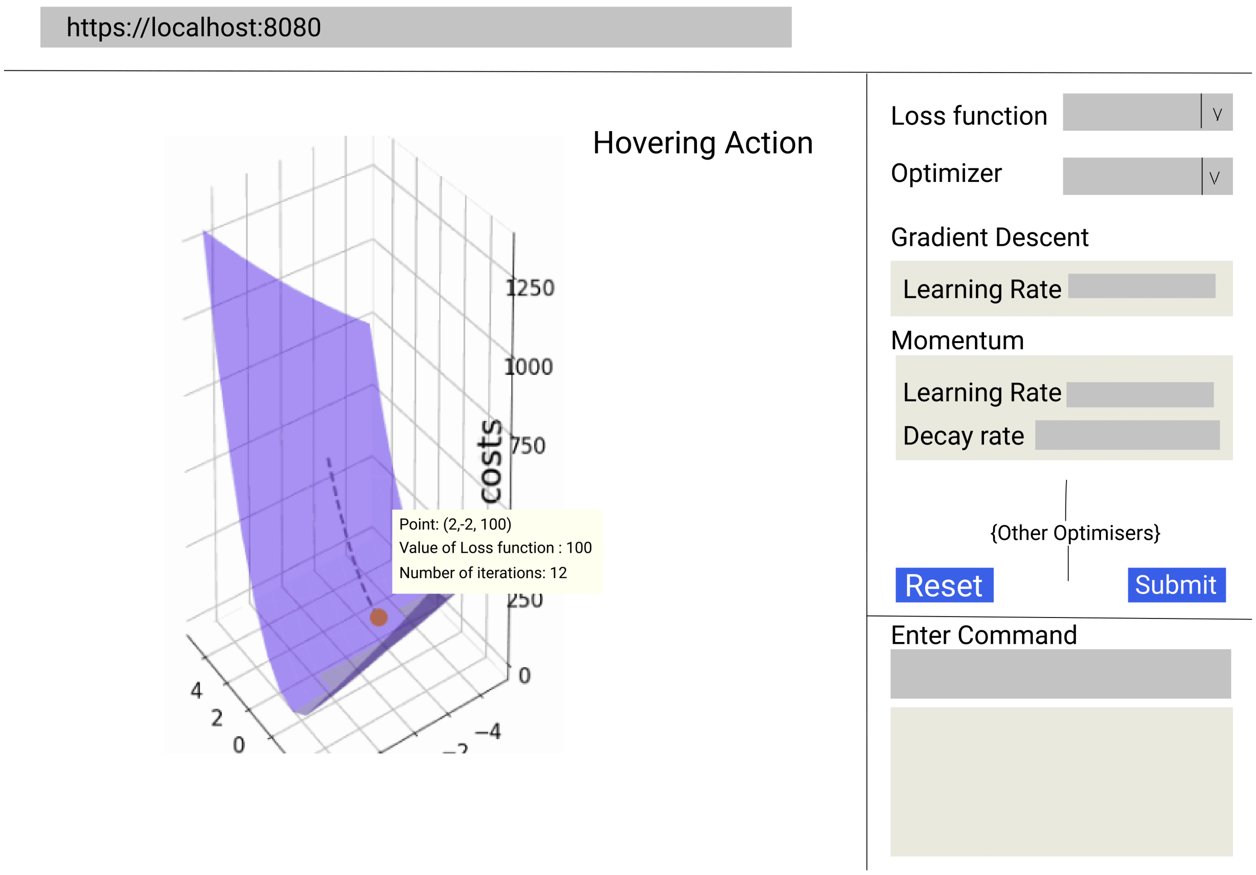


The right column contains the user given parameters and each of the optimizers has it's own list of hyperparameters to be given. Since, the SRS asks to select one optimizer at a time, giving more than one will lead to ignoring the data passed with that since the optimizer will already be selected as some single specific value and only the hyperparameters related to that will be read in the programs ahead. Since the diagram is just an overview and not perfect, there are some things left out like there will be a field for specifying number of iterations as well in the right column.

This is an example of the window that the user is seeing after the gif has been rendered. There can be points marked after every step but this window is just for the sake of demonstration as to where the gif will appear and how it will be rendered, though the exact nature of the graph is open to changes according to SRS.



This window shows how the hovering action is supposed to work. The mouse cursor is assumed to be near the orange ball and the status of the gradient descent is shown in the box near the ball.

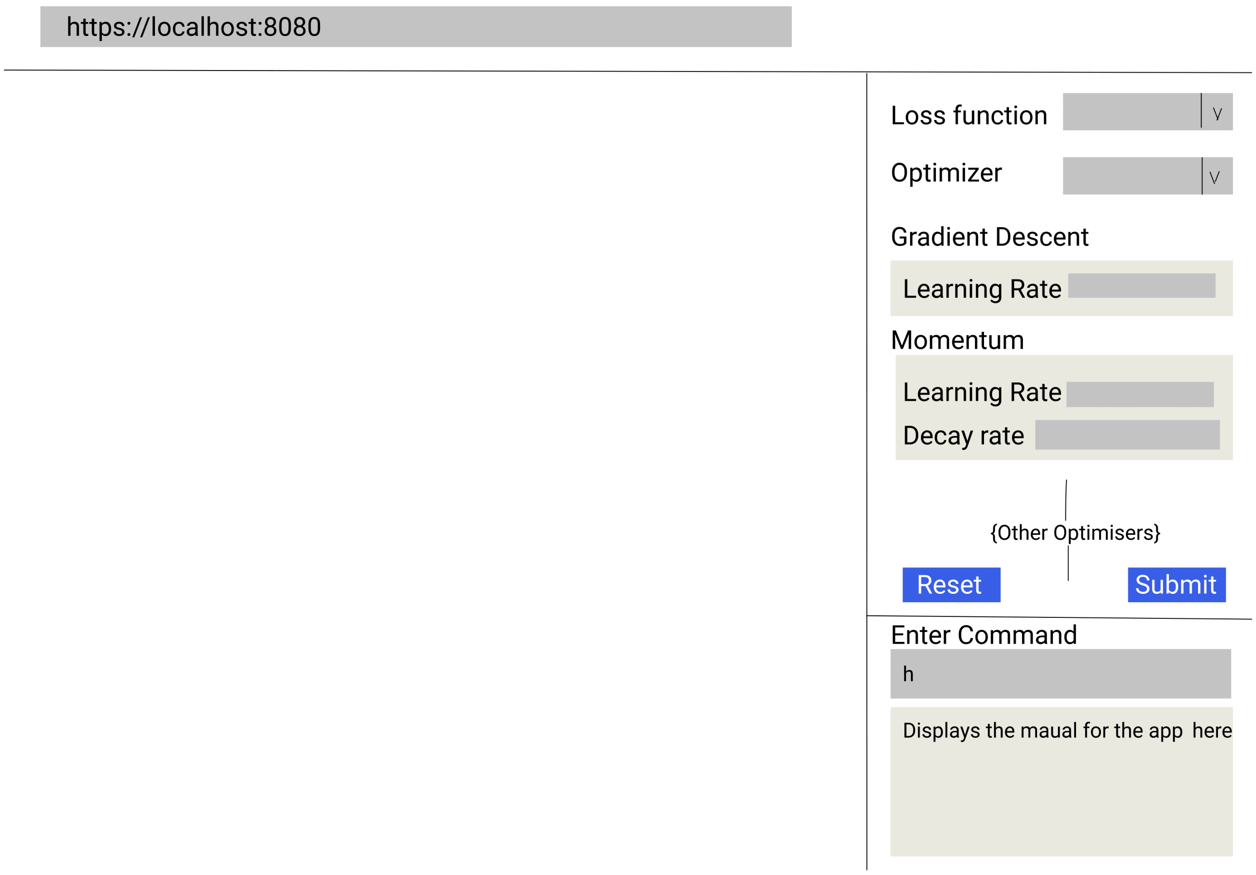

https://localhost:8080

Hovering Action

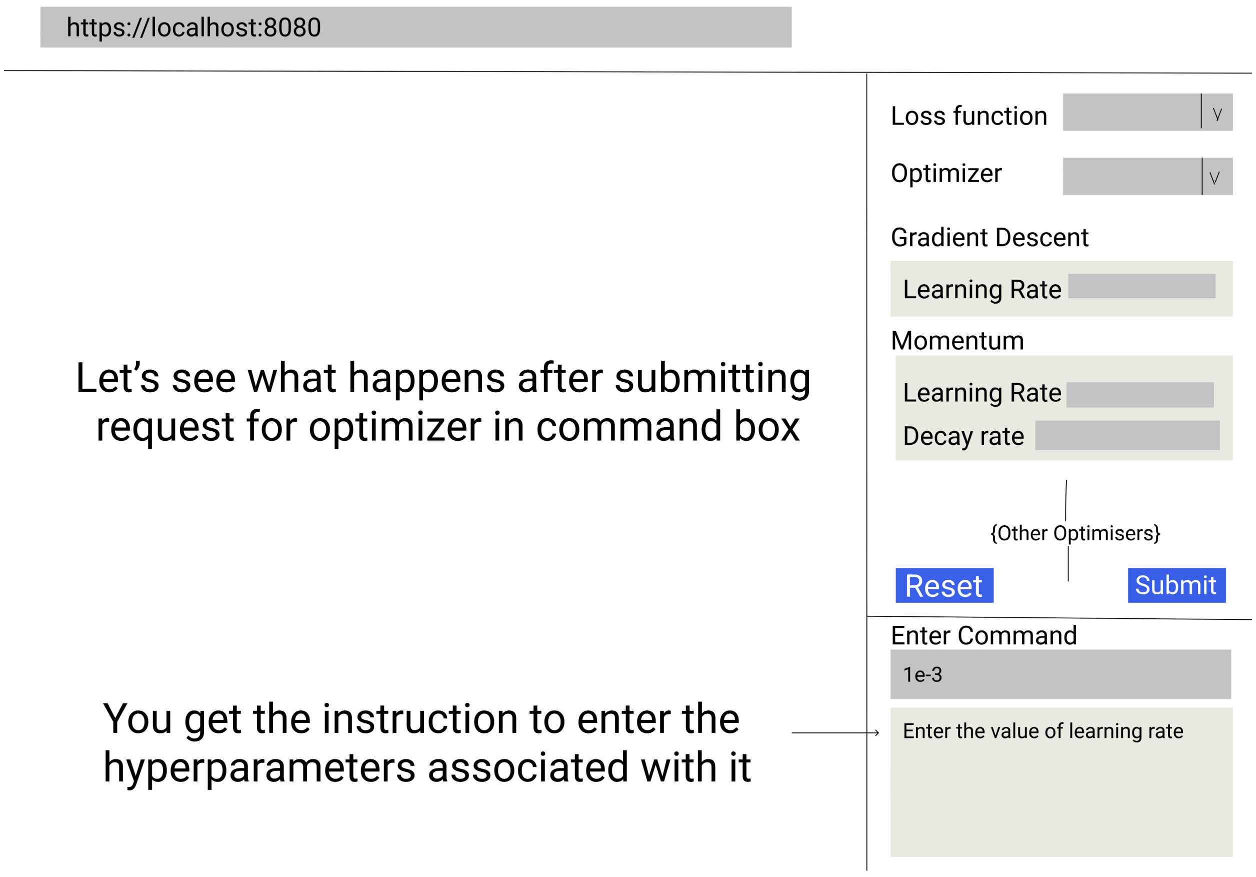
Point: (2,-2, 100)
Value of Loss function : 100
Number of iterations: 12

Now we will see the rendering of the command box which is on the bottom of the right column. That portion is divided into two parts, a box to enter the command and another bigger box to give the further instructions after typing the command and pressing enter. We will see the rendering of the command box next.

This image shows the result of typing the command for 'help' i.e. 'h' in the command box and the result is displayed as the manual for the webapp in the bigger box. The bigger box is supposed to have scroll bar on the right side as well which is not detailed in the shown wireframe.



The next image explains what to do when an optimizer is entered in the command box. The bigger box will give instructions on what to do next to the user.



1e-3

Enter the value of learning rate

Let's see what happens after submitting request for optimizer in command box

You get the instruction to enter the hyperparameters associated with it

6.3 Screen Objects and Actions

In this section we will discuss the different screen objects and actions associated with those objects. Almost every component has been extensively discussed before in this document. We will just discuss the functionality of the command box, specifically the bigger one. It is just there to tell the user how to use the command box. For instance, we can provide a message like enter command you want to run in the command box and type 'h' for help.

The result of a particular command is also rendered on this small box. For instance when typing for a command which requires further action from user's side, it will ask for the user to complete the action. For the command for choosing the optimizer the result has been shown in one of the previous images. For the other commands like choosing functions it will open the drop down list first by interacting with the container of the form, and then will ask the user to type one of the functions from the drop down list. After completion of a command the box displays the message that the command is completed.

7 Requirement Matrix

Requirement	Satisfied by
Web app compatible with all browsers and machines	**setup** module and **Windows** module
Interaction through GUI	Frontend libraries in **Windows** Module
Fast website	Using fast and light libraries and frameworks like numpy and pytroch-cpu
Implementing shortcuts	Command section and detect_key_press function in **Windows** module
Nature of data	Refer to the Data design section
Backend running on CPU	Use of pytorch-cpu
Visualisation	Use of light visualisation libraries in **Plot_area** module
Taking in values from user	Right column of the webpapp rendered using **Windows** module
Computations done in user's hardware	Hosting the server on user's machine using ExpressJS.
Running SGD in backend	Using fast and light libraries and frameworks like numpy and pytroch-cpu in **Gradient_descent** module
Connecting points generated during SGD	**generate_figure** function in **Plot_area** module
Hovering action	**hover_action** function in **Windows** module
Memory Constraints	All of the used libraries are light in size maximum being 85MB
Validating user given data	Checks used in **Windows** module
Security Requirements	Sanitisation in frontend using **Windows** module and in backend using **Gradient_descent** module
Modularity of code	Refer to Design Rationale section 3.3
Portability	Use of OS independent libraries/frameworks like ExpressJS, numpy, pytorh-cpu.

8 Timeline

Most of the time will be devoted to completion of the **Windows** module. Besides, it others are mostly independent and the hence the work can be completed parallelly. The complete timeline(expected) is shown below.

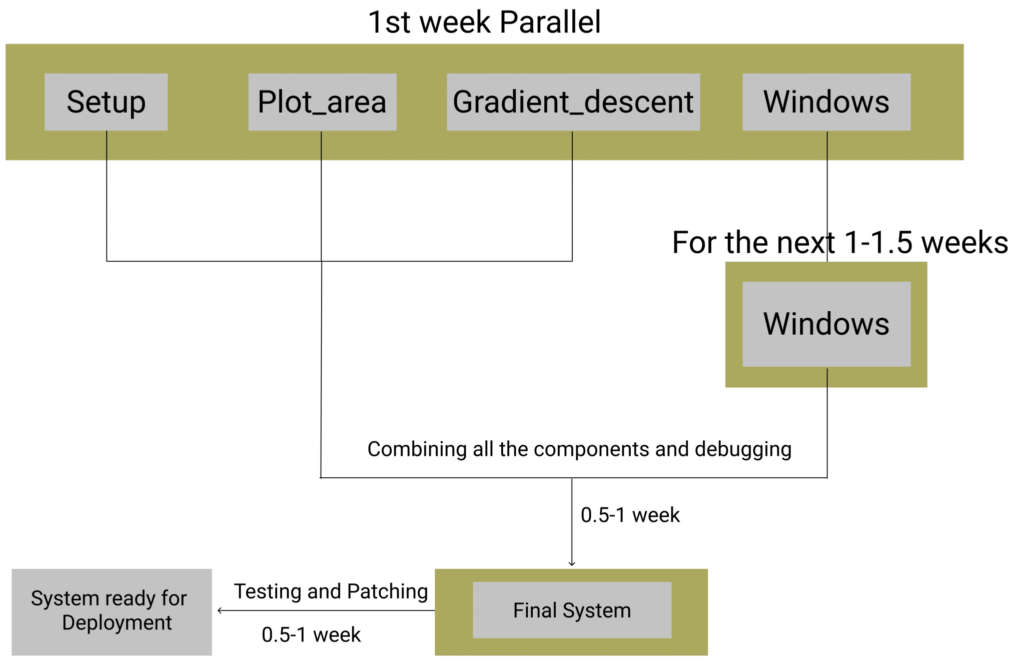


Figure 3: Timeline for completion of project

9 Appendix

9.1 Stochastic Gradient Descent

We take a initial point x_0 and a function f and and a learning rate. We successively update x_0 as follows

$$x_{i+1} = x_i - \eta \cdot \nabla_{x=x_i} f(x)$$

here η is called the learning rate, which is used to regulate the step size that we take. This is the crude version of Stochastic Gradient Descent Algorithm.

It can be run with various optimizers. The interested ones are suggested to check [1] for how they work.

References

[1] An overview of gradient descent optimization algorithms. https://ruder.io/optimizing-gradient-descent/.