MATH 211 Final Project

Visualizing Loss Functions

Connor McGeehan

December 2025

1 Introduction

There are many applications of Multivariable Calculus to almost any field that you could think of. For my final project, I wanted to combine my experience with computer science and data science to show one way that calculus is used when creating machine learning models.

To do this, I decided that I would create a relatively basic program that visualizes a commonly used loss function, the Mean Square Error (MSE) loss function, it's gradient at different weights, and the level curves of the function. To make the visualization more interesting, I added a quartic penalty term. The user can adjust the value of alpha for this term, so that they can see how the gradient, function surface, and level curves change with different penalties.

This report details the steps that I took to calculate this function, manipulate the data, and create the visualization while creating the program.

2 Data Description

The data for this project was collected for a final project that I am doing for a data science course I am currently taking. The data contains some basic information about Division III Cross Country Runners who competed at the National Meet in 2021, 2022, or 2023, and was collected via the API for LACCTiC.com. The columns that I used for this project were Season Record, Personal Record, and Number of Races Run. These were chosen

somewhat randomly, but I did find that when they were standardized they created a nice visualization.

Overall, the specific data does not play an extremely important role in this project, but I thought it would be important to reference it's origin and meaning.

3 Loss Function Definition

3.1 The Loss Function

The loss function used in this project is a MSE loss function with a quartic penalty. I added the quartic penalty because it alters the loss function in interesting ways, and as a result the project is much more interesting. The full loss function can be seen below:

$$L(w_1, w_2) = \frac{1}{n}\sum_{i=1}^{n}\left((w_1 X_{1i} + w_2 X_{2i}) - y_i\right)^2 \; + \; \alpha(w_1^4 + w_2^4)$$

A MSE Loss Function is very common in data science modeling. It is used to calculate the error of a model for specific weights. Algorithms for creating models, such as linear regression, aim to minimize this loss function to find the values of weights that predict an outcome most accurately. This is a fantastic display of how data science relies on calculus, the creation of a predictive model relies on the optimization powers of calculus.

3.2 The Quartic Penalty

For this project I added a quartic penalty. Penalties are added to loss functions to aid the process of model selection. Some commonly used examples are LASSO and Ridge regularization. These are two common methods to help eliminate unneeded predictors, or decrease the size of some of the weights. The quartic penalty specifically punishes large weights by adding the quartic value of each weight to the result of the MSE. This method is not as commonly used in the real world, as it is a very extreme punishment. However, for this project it provides some very interesting visualizations.

3.3 Variables

There are many different variables used in this equation, and their meaning is necessary to fully understand what is happening with the loss function.

n The number of observations in the dataset

w_1 The weight for the Season Record observation

$X_1 i$ The Season Record observation i

w_2 The weight for the Number of Races Run observation

$X_2 i$ The Number of Races Run observation i

y_i The Personal Record observation i, the result being predicted

α The regularization parameter controlling penalty strength

3.4 Gradient of the Loss Function

The gradient of the loss function is useful for determining the minima of the function. The gradient is a vector that points in the direction of greatest increase in a function, it is obtained by taking partial derivatives of the function. This allows for the process of gradient descent, which will be discussed in the next section. The general formula for the gradient is below:

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1} \frac{\partial f}{\partial x_2} \cdots \frac{\partial f}{\partial x_n} \right)$$

As seen, the gradient is simply the partial derivative of f with respect to each parameter. This means that the gradient of our loss function will be taken with respect to w_1 and w_2, as those are the variables that we are interested in measuring the change of. This results in the following equation:

$$\nabla L(w_1, w_2) = \begin{pmatrix} \frac{2}{n} \sum_{i=1}^{n} \left((w_1 X_{1i} + w_2 X_{2i}) - y_i \right) X_{1i} + 4\alpha w_1^3, \\ \frac{2}{n} \sum_{i=1}^{n} \left((w_1 X_{1i} + w_2 X_{2i}) - y_i \right) X_{2i} + 4\alpha w_2^3 \end{pmatrix}$$

4 Optimization and Interpretation of Critical Points

4.1 Critical Points

The critical points of a loss function are *critical* to data science. Critical points are places where the gradient of a function is equal to 0. For models in $\mathbb{R}^3$, a gradient equaling 0 means that there is either a local minima, maxima, or saddle point. Local minima and maxima are where the function either increases or decreases, respectfully, in all neighboring directions. A saddle point is where the function may increase in some directions but decrease in others around the point.

For loss functions we are interested in locating the local minimum, this point is where the error of the data for two specific weights is the lowest. Most simple MSE loss functions are convex, meaning that they have some sort of bowl shape to them. This also means that there is a single minimum on the function, which is the ideal combination of weights for the model. Some models, such as neural networks, can produce different results with loss functions, and this can lead to there being multiple minima. When this is the case each critical point needs to be identified and then tested to determine which one has the lowest error. However, for our equation there is only one minimum.

There are two main methods for locating critical points for a loss function: setting the gradient equal to zero and then classifying the results, and gradient descent. The two are discussed briefly below.

4.2 Gradient Equal to Zero vs. Gradient Descent

Setting the gradient equal to zero is the most accurate way to identify critical points. With a little bit of algebra you can determine every point where the gradient is equal to zero, and as a result you are left with a list of critical points. These points can then be plugged into the original equation, and whichever one has the lowest value is your local minimum. However, it is not always easy to set the gradient equal to zero. Some functions are too computationally intensive to be worth this method. This is where gradient descent comes in.

Gradient descent is an algorithm where the gradient is calculated at a random starting point on the functions surface, and then a new point is

obtained by travelling in the opposite direction of the gradient. This method aims to follow the steepest path down a loss function by taking the opposite of the gradient which points in the direction of the greatest incresase. Gradient descent does not require nearly as much computation as setting the gradient equal to zero, making it much more efficient for complex functions. This method is not without pitfalls either, though. It is not as precise as setting the gradient equal to zero, as there is a set amount that you travel along the gradient. This method will get you close to the minimum of a function, but it is not guaranteed to find the true minimum's location.

Both of these methods also suffer from the fact that they can miss the true global minimum by getting trapped in a local minimum. These are places where there is a critical point where it is a minima, but it is not the lowest point of the function. This issue can be addressed for both methods (increasing the range of values being checked when setting gradient equal to zero, and introducing randomness or adjusting step size for gradient descent), but neither can be completely avoided. However, this is not an issue for the loss function being examined in this project, as it is still convex.

5 Model Limitations

There are a few key limitations to this specific model that are results of choices made to highlight some of the calculus applications instead of typical data science decisions. As mentioned earlier, the quartic penalty is not a commonly used method of regularization. It is mathematically interesting, as the functions surfaces changes dramatically when it is added, but it is not practical in most models. The sudden growth that results from large weights makes it hard to use practically.

The data for this model was also normalized with a z-score, which changes the meaning of the weights. Due to the drastic difference in values of the Number of Races Run column and the Season Record column, I used a z-score to normalize the data, meaning that both columns had a mean of 0 and a standard deviation of 1. This made the visualization much clearer; beforehand it looked like a long valley due to the difference in impacts of changing the weights slightly. With this change, the weights now show the impact of each variable, so they can see which variable is more impactful by observing which weight is larger. However, you can no longer use the weights to say "a one second change in season record leads to a X increase in personal

record."

6 Implementation

This project was created with python, primarily utilizing the matplotlib library. I will briefly discuss some of the decisions I made while implementing this project, but if you are interested in seeing more all of the code can be viewed in the GitHub repository: https://github.com/ConnorMcGeehan/calcIII-visualizations/tree/main

In order to make this project easy to use and update, I utilized many different classes to control small aspects of the visualization. The `BasePlot` class houses most of the information about the data and the functions. It pulls the data out of the csv, and then calculates what the loss would be for each combination of weights on the set axes. This class also computes the gradient for the set values of `w1` and `w2` values, which indicate the values of the weights for the Season Record and Number of Races Run columns. The `FunctionSurface` and `LevelCurves` house the code needed to plot the respective plots given the information in `BasePlot`. The code that controls the sliders is inside the `UI` class, which also actually shows the plots. This class is also responsible for updating the plots when the sliders are changed.

I chose to add the sliders to help assist with seeing how the concepts learned in calculus actually relate to data science. By being able to see how the gradient changes at different weights, you can visualize how gradient descent would be implemented. You can also adjust the graph itself by changing alpha, which allows for more interesting surfaces to be visualized than a basic convex MSE loss function.

7 Visualizations

The plots open up with the weights and alpha all set to 0. This shows what the MSE loss function looks like in a normal state. As seen in Figure 1, the MSE loss function is a very basic convex function. It has one local minimum, and increases fairly evenly in all directions. The gradient is highlighted with a red arrow on both plots, although it is more easily visualized with the level curves.

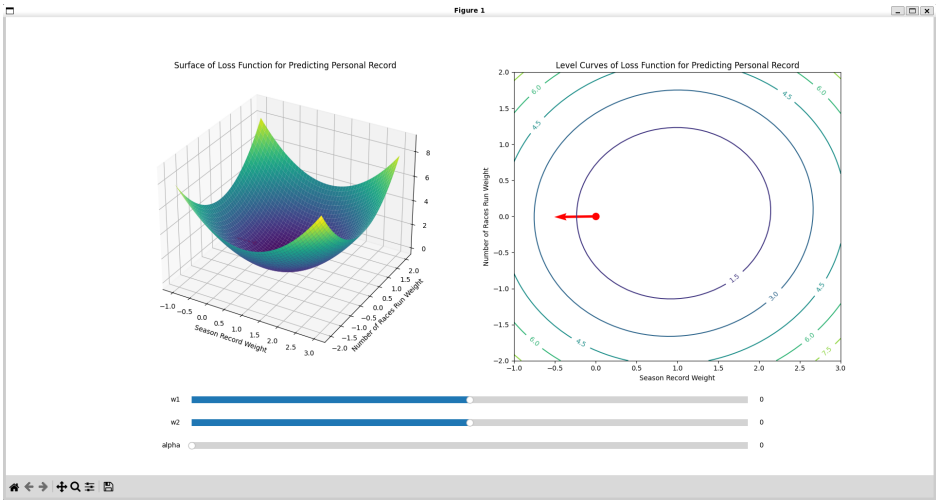
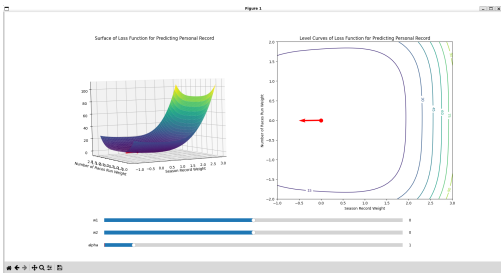
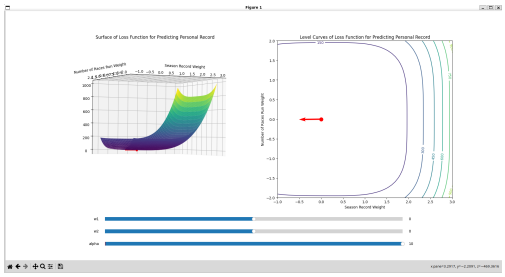


Figure 1: Original loss surface visualization

Adjusting the alpha slider allows the user to see what happens when a quartic penalty is added. Although the appearance of the graph doesn't change much between $\alpha = 1$ and $\alpha = 10$, you can see on both the z-axis and the level curves how the amount of loss changes with the values of alpha. As seen in Figure 2a and Figure 2b, when the quartic penalty is added, the function becomes much flatter around the local minimum, but incredibly steep once it does start to increase. This makes a lot of sense when considering what the quartic penalty is doing: setting each weight to the fourth power and then multiplying them by α. This is another great example of why quartic penalties aren't always a very practical option.



(a) Setting $\alpha = 1$



(b) Setting $\alpha = 10$

Figure 2: Effect of quartic penalty with different α values

Figure 3 allows you to see what happens when you adjust the weights so that they are at the minimum value of the function. For the MSE loss function, the minimum is around $L(1, 0)$. When the gradient is adjusted to that location, you can see that it disappears, indicating that it is at a critical point.

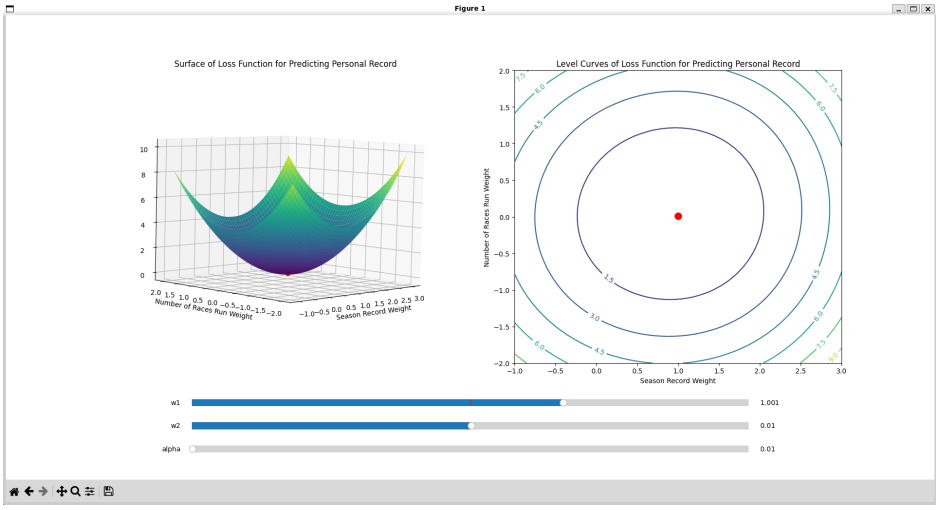
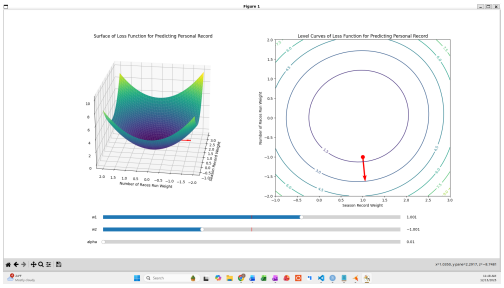
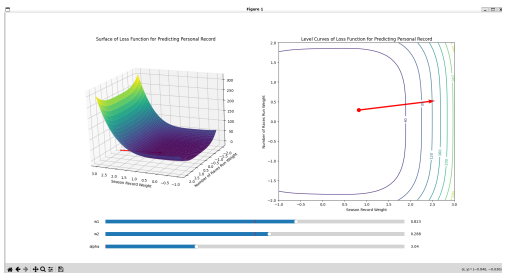


Figure 3: Gradient at the Local Minimum

Finally, Figure 4a and Figure 4b allow you to see what the plots look like with varying weights and alpha configurations.



(a) Changing the weights



(b) Changing the weights and alpha

Figure 4: Loss surfaces with varying weight and penalty configurations

8 Conclusion

This project covers many different areas of multivariable calculus, and demonstrates how crucial calculus is to data science, computer science, and anything that we do that is impacted by predictive modeling. Without the optimization techniques that are made possible through calculus, it would be next to impossible to efficiently create predictive models.

By restricting the MSE loss function to two variables, I was able to explore the geometry of the function both mathematically and visually. Rather than relying on algebra and notation to describe how the function was behaving, I was able to create plots that show the user exactly what is happening when they do things like computing the gradient and change penalty terms.

These visualizations have a great tie-in to my data science course, as they provide a clear image of what happens whenn you add a penalty term to your loss function. While the original MSE loss function is a simple convex surface, adding the quartic penalty drastically altered the geometry of the figure. It was also very interesting to visualize how the gradient shrunk when it approached the local minimum, which aids in the understanding of gradient descent.

Overall, I believe that this project highlights the value of using visualization to understand multivariable functions, their gradients, and their level curves. By being able to view how different parameters alter different parts of the plots, the user can gain a new understanding and appreciation for how calculus can be applied to data science and many other areas.