Mathematical & Numeric Optimization

Matt Drury & Cary Goltermann

March 1, 2017

Introduction

These notes will discuss the intuitions of numeric optimization via gradient descent. This intuition will be build via mathematical inspection of optimizable functions, examples of the coefficients for linear and logistic regression will be discussed at length. In addition, potential problems with the gradient descent algorithm will be pointed out along with some straightforward solutions; again, this is all in the pursuit of build intuition for the process.

Optimization

Optimization is the study of methods for finding the maximal of minimal value of a function.

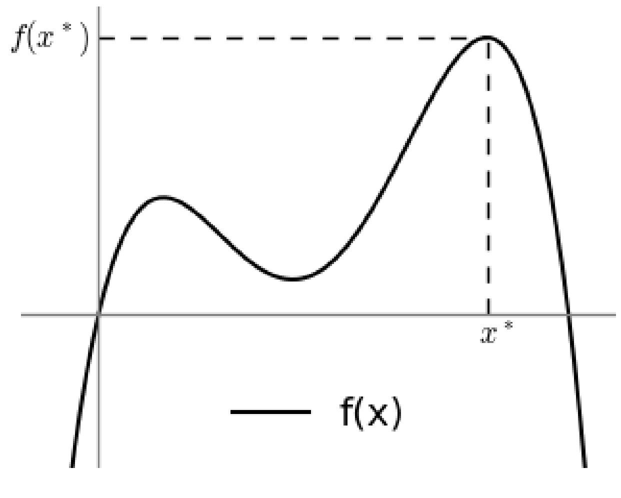

Figure 1: One Variable Function

Figure 1 above plots a function of one variable, in it we see x^*, denoting the value of x that yields the maximal value for $f(x)$, or the *argmax* of f. If we were to optimize over this function the end goal would be to find the $x*$, the *argmax* of f, thus providing us with knowledge about how to get the "most" out of f.

In figure 2 we can see the contours of a two-dimensional function plotted. Contour lines have the property that the value of the function is the same at every point along the same line. These lines also go by the name level curves. Again we can see the location in this space that yields the maximal value along this two-dimensional function, the *argmax*, labeled as x^*.

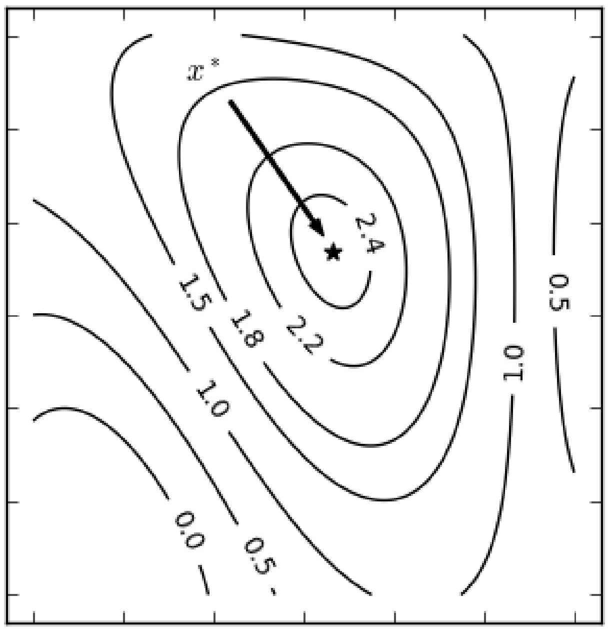

Figure 2: Two Variable Function

Examples

1. **Linear Regression**

   In linear regression we are trying to fit a line to our data, $\boldsymbol{X}$, in so doing finding optimal β's that minimize the sum of squared errors.

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \sum_{i=0}^{n} \Big(y_i - \sum_{j=0}^{p} \beta_j X_{ij}\Big)^2$$

This quantity sums over all the features in the data, $0 - p$, and all the data points in the set, $0 - n$. We can also express this with vector notation to reduce some of the indexing clutter.

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|$$

2. **Logistic Regression**

   In logistic regression we use the logistic, or sigmoid, $\sigma(x)$, link function to make probability predictions after taking the same linear combination we saw above in linear regression, $\boldsymbol{X}\boldsymbol{\beta}$.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

   We are interested in minimizing the total cross-entropy,

$$H(p, q) = -\sum_{x} p(x)\, log(q(x)),$$

   of these probabilities with the true labels from the set $\{0, 1\}$ over all the data points.

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad -\sum_{i=0}^{n} y_i\, log\big(\sigma(\boldsymbol{X}_i\boldsymbol{\beta})\big) + (1 - y_i)\, log\big(1 - \sigma(\boldsymbol{X}_i\boldsymbol{\beta})\big)$$

   where

$$\boldsymbol{X}_i\boldsymbol{\beta} = \sum_{j=0}^{p} \boldsymbol{X}_{ij}\boldsymbol{\beta}_j$$

   .

Constrained Optimization

Sometimes optimization problems are constrained. In these cases we are interested in the maximum/minimum value of f but only in the cases that satisfy some constraints. An example constraint might be that we want the magnitude of the values in each dimension to be less than or equal to a certain threshold. We could represent such a constraint in two dimensions as such:

$$x_1^2 + x_2^2 \leq M^2$$

And so the optimization problem we would be facing could be written: maximize $f(x_1, x_2)$ subject to the constraint $x_1^2 + x_2^2 \leq M^2$.

This constraint can also be seen visually. In figure 3 below we see the same contour plot as above; however, now there is an added gray region with radius M representing our constraint. The constraint region in this example is circular due to the square terms in the above constraint.

We can see that the constrained maximum of f, x^*_{const}, occurs at the first point that the constraint region touches a contour line of f.

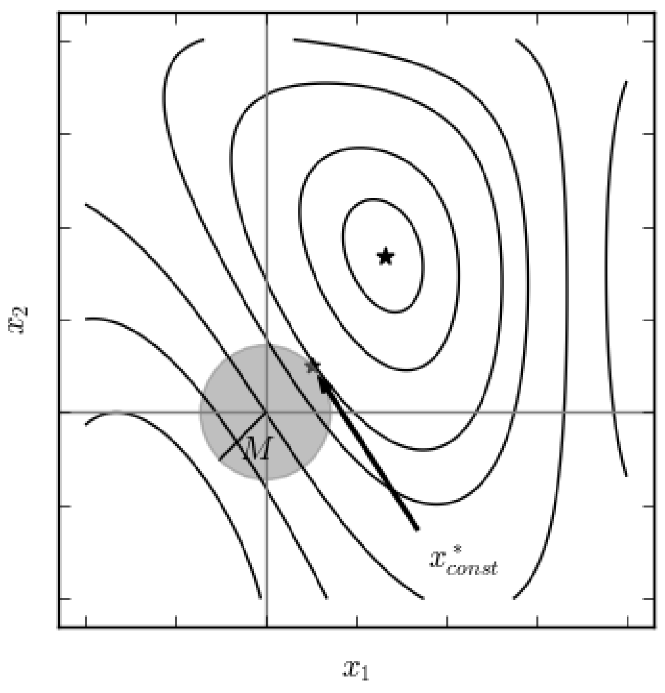

Figure 3: Constrained Two Variable Function

Example

Regularized Linear Regression

Sometimes, in linear regression, we want to control the magnitude of the betas that are learned by our model with the intuition that unrestrained betas can lead to overfitting. To constrain our betas all we simply add a term to our cost function summing up the squares of the betas and scaling that by some factor λ.

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \sum_{i=0}^{n} \big(y_i - \sum_{j=0}^{p} \beta_j X_{ij}\big)^2 + \lambda \sum_{j=0}^{p} \beta_j^2$$

The magnitude of λ is tuned as a hyperparameter with the intuition that larger values of λ will make the effect of $\sum_j^n \beta_j^2$ larger, thus incentivizing smaller betas.

There is another, equivalent, way we can express this optimization problem; as a constrained form of the linear regression cost function.

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \sum_{i=0}^{n} \bigl(y_i - \sum_{j=0}^{p} \beta_j X_{ij}\bigr)^2$$

subject to the constraints:

$$\sum_{j=0}^{p} \beta_j^2 \leq M.$$

Legrange Multipliers

This is representative of a general pattern:

Every constrained optimization problem has an equivalent unconstrained optimization problem.

The process of moving from the constrained problem to the unconstrained problem is called Legrange Multipliers.

The Gradient

We want to develop methods for solving optimization problems, and all methods start with a common concept, the gradient.

Definition

Suppose that $f(\beta_0, \beta_1, \beta_2, ..., \beta_p)$ is a function of multiple arguments. The gradient of f is the vector of the partial derivatives with respect to each of f's inputs.

$$\nabla f(\beta_0, \beta_1, \beta_2, ..., \beta_p) = \left(\frac{\partial f}{\partial \beta_0}, \frac{\partial f}{\partial \beta_1}, \frac{\partial f}{\partial \beta_2}, ..., \frac{\partial f}{\partial \beta_p} \right)$$

Note, this vector depends on the point $(\beta_0, \beta_1, \beta_2, ..., \beta_p)$, this is why the gradient is sometimes referred to as a vector field.

Similarly to the derivative the gradient corresponds to the slope of hyperplane tangent to the function.

Properties

- The gradient at a point is perpendicular/orthogonal to the contour line that runs through that point, see figure 4.
- At a non-maximum/minimum, the gradient points in the direction that the function increases most quickly.
- At a maximum/minimum (optima), the gradient is the zero vector.

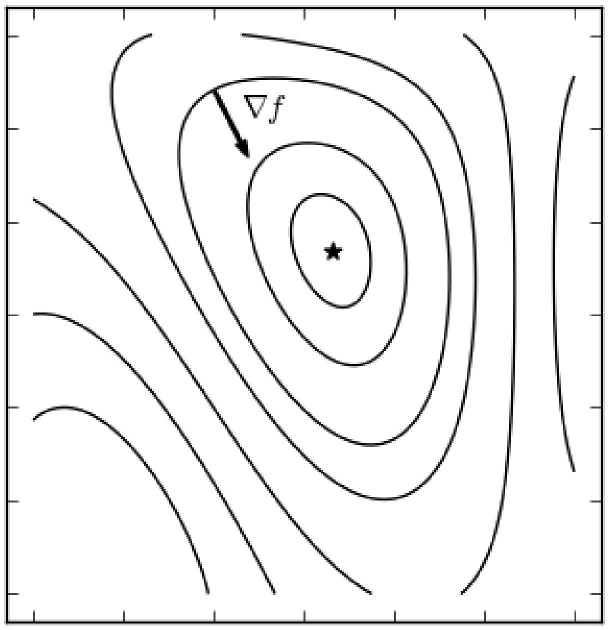

Figure 4: Gradient for Two Variable Function

Using the Gradient to Find Optima

Solving Explicitly

Very rarely, the following algorithm will result in a closed form solution to finding an function's optima:

1. Write down the function you want to optimize.

2. Take all of the necessary partial derivatives to calculate the gradient of the function by hand. This sometimes fails.

3. Set the gradient equal to the zero vector, and solve the resulting system of equations. This often fails.

Example

1. Function: $f(\beta_1, \beta_2) = \beta_1^2 - 2\beta_1\beta_2 + 3\beta_2^2 + 4\beta_1$

2. The partial derivatives:

$$\frac{\partial f}{\partial \beta_1} = 2\beta_1 - 2\beta_2 + 4$$
$$\frac{\partial f}{\partial \beta_2} = -2\beta_1 + 6\beta_2$$

   $\Rightarrow$ Gradient: $\nabla f(\beta_1, \beta_2) = \big(2\beta_1 - 2\beta_2 + 4, -2\beta_1 + 6\beta_2\big)$

3. Set gradient equal to zero vector, yields system of equations:

$$\begin{aligned} 2\beta_1 - 2\beta_2 &= 4 \\ -2\beta_1 + 6\beta_2 &= 0 \end{aligned}$$

   This system can be solved with scipy and numpy: $scipy.linalg.solve(\left[\begin{smallmatrix} 2 & -2 \\ -2 & 6 \end{smallmatrix}\right], \left[\begin{smallmatrix} -4 \\ 0 \end{smallmatrix}\right])$. The solution is at $(-3, -1)$, which is where the minimum of the function occurs.

Example: Linear Regression

If we apply the same procedure to the linear regression problem we get the result:

$$\hat{\beta} = scipy.linalg.solve(X^T X, X^T y),$$

which is often written as:

$$\hat{\beta} = (X^T X)^{-1} X^T y,$$

known as the normal equations.

Some people say that it's a miracle the ordinary least squares linear regression results in a closed from solution for the beta coefficients. As such it's important to know that linear regression exists as the exception, not the rule. In fact, the coefficients of logistic regression **can not** be solved for in this way.

Gradient Descent

If explicit solutions often fail, how can we still optimize an arbitrary function? To answer this question we return to the defining property of the gradient at a point.

> *The gradient points in the direction which the function increases most quickly.*

With this definition in mind, consider the tactic of repeatedly walking in the direction of greatest increase, the gradient, or negative of the gradient for decrease.

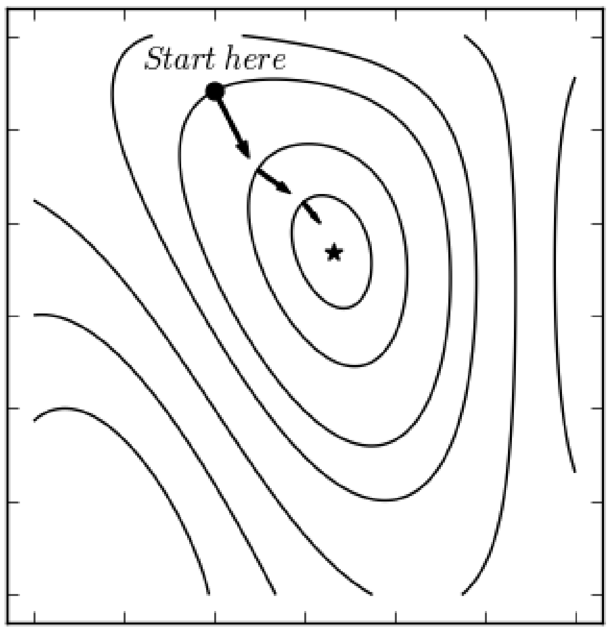

Figure 5: Gradient Descent for Two Variable Function

Algorithm

There are two main parts to performing gradient descent.

Pre-Algorithm

- Write down the function to optimize.
- Calculate the gradient analytically.

Pseudocode

```
• Choose an initial point $x_0$.
• Until stopping criteria met:
    1. Plug current point into gradient. $\nabla f(x_0)$.
    2. Move in the take a step along the gradient:
        • $x_{i+1} \leftarrow x_i + \alpha \nabla f(x_0)$      gradient ascent
        • $x_{i+1} \leftarrow x_i - \alpha \nabla f(x_0)$      gradient descent
    3. Repeat.
```

The learning rate, α, corresponds to how far along the gradient vector you step each time. It is set to a positive number less than one, so as to guarantee we don't take too large a step and possibly "overshoot" our goal.

Stopping Criteria

There are many ways that one can control how long you optimize for, the simplest is only iterating for a predetermined number of steps. There are, however, more sophisticated means of control.

1. When $\|\nabla f\| < \epsilon$, where ϵ, the convergence threshold, is a small positive number. I.e. when the length of the gradient is small, almost the zero vector.
2. When $\frac{\|f(x_{i+1}) - f(x_i)\|}{\|f(x_i)\|} < \epsilon$. I.e. when the percentage update is small.

Trouble Shooting

Local Optima

When the function has multiple local optima, gradient descent will converge to only one of them. In this case you may not end up anywhere near the global optima. An illustration of this can be seen in figure 6.

Solutions

- Many common problems, e.g. linear/logistic regression, have exactly one local optima, so this issue does not arise. Functions like these are called globally convex.
- Add an inertia term to your descent to "power through" getting stuck in a non-optimal local optima.

- In general, there is no silver bullet, you must add some randomization to your algorithm to try and get to better optima given different starting locations.
    - Start at many random points. Choose the best optimal value from all that are found.
    - Add some random noise to the gradient.

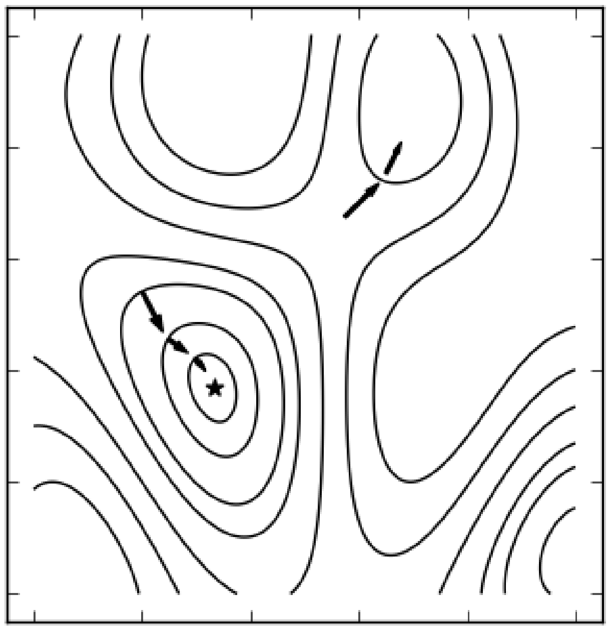

Figure 6: Gradient Descent Starting from Different Initializations

Poorly Scaled Features

In regression problems many times different features are measured on very different scales. e.g. measuring pounds a person weighs vs. height in feet. The other way we can see this problem is by realizing that whatever solution we come to should be invariant under constant transformations of our data. e.g. instead of measuring height in feet, converting everything to inches.

In this situation the level curves of the loss function become very long and thin, see figure 7. The ramification is that the gradient, which points in the direction of greatest assent in proportion to the rate of that change will predominantly point in the directions of the features that are on a larger scale.

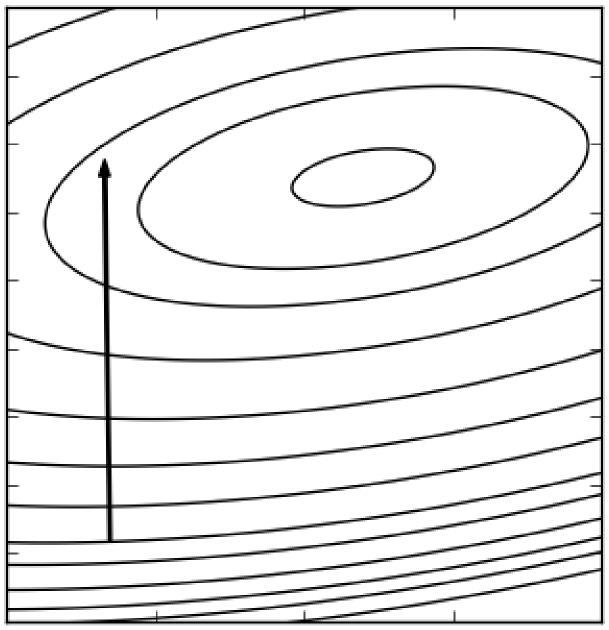

Figure 7: Poorly Scaled Features

Solution: Scale the features!

Replace $\boldsymbol{X}_j$ with:

$$\frac{\boldsymbol{X}_j - mean(\boldsymbol{X}_j)}{std(\boldsymbol{X}_j)}.$$

This will make all of the features have mean zero and variance one. Graphically the level curves after standardizing are much more round.

Stochastic Gradient Descent

The cost function of many important machine learning algorithms are sums of a simpler function. These sums operate over all of the observations in our data set.

With this in mind, we can consider a computationally cheaper way to perform gradient descent by only considering what the gradient of our cost function is with respect to a single data point. This flavor of gradient descent is called stochastic gradient descent, SGD, because we're adding some randomization to our descent path by only considering a single point for each update. The intuition here is that most of the time, though not always, the gradient of our loss function with respect to a single data point will generally point in the correct direction.

Linear Regression

Below we can see the equations for the linear regression loss with respect to an entire data set, $L(\boldsymbol{\beta}, \boldsymbol{X}, \boldsymbol{y})$, and with respect to a single data point, $l(\boldsymbol{\beta}, \boldsymbol{x}, y)$.

$$L(\boldsymbol{\beta}, \boldsymbol{X}, \boldsymbol{y}) = \sum_{i=1}^{n} \left(\boldsymbol{y}_i - \sum_{j=0}^{p} \boldsymbol{\beta}_j \boldsymbol{X}_{ij} \right)^2$$
$$l(\boldsymbol{\beta}, \boldsymbol{x}, y) = \left(y - \sum_{j=0}^{p} \boldsymbol{\beta}_j \boldsymbol{x}_j \right)^2$$

Visualizing Stochastic Gradient Descent

The effect of performing updates like this can be seen in figure 8 below. We see that the path is not very direct, not necessarily moving in the "best" direction, sometimes moving in the opposite way. But in aggregate it moves towards the optima quite well.

This happens because, with respect to a single data point, the "best" direction is not the same as the global "best" at that location for the betas at that iteration.

In fact, we can see that the expectation of the gradient of the loss function with respect to a single data point is the average gradient.

$$E[\nabla l(\boldsymbol{\beta}, \boldsymbol{x}, y)] = \sum_{i=0}^{n} \frac{1}{n} \nabla l(\boldsymbol{\beta}, \boldsymbol{x}, y) = \nabla \frac{1}{n} \sum_{i=0}^{n} l(\boldsymbol{\beta}, \boldsymbol{x}, y)$$

This works because of the linearity of expectation and derivatives.

Why is Stochastic Gradient Descent Desirable

Sometimes we don't have access to all of the data!

- Huge datasets cannot fit into RAM.
- Our data is streaming to us, and we only have access to data temporarily.
- Our data is streaming to us, and we want to import our model in real time. This is called online learning.

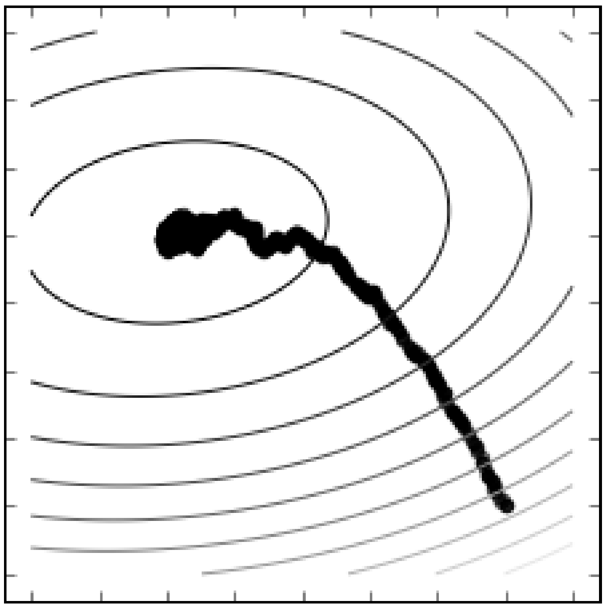

Figure 8: Stochastic Gradient Descent Path

In addition, frequently we have to perform much fewer iterations, and therefor, calculations to converge to an optima. And getting computational gains is never a bad thing.

Comparing Gradient Descent with Stochastic

- **Gradient Descent**

  Updates, until convergence: $\boldsymbol{\beta}_{i+1} \leftarrow \boldsymbol{\beta}_i + \alpha \, \frac{\partial}{\partial \boldsymbol{\beta}_i} L(\boldsymbol{\beta}_i, \boldsymbol{X})$.
  Here $\boldsymbol{X}$ is the entire training data set.

- **Stochastic Gradient Descent**

  Updates, until convergence: $\boldsymbol{\beta}_{i+1} \leftarrow \boldsymbol{\beta}_i + \alpha \, \frac{\partial}{\partial \boldsymbol{\beta}_i} l(\boldsymbol{\beta}_i, \boldsymbol{x})$.
  Here $\boldsymbol{x}$ is a single observation from the training data set.

Deriving the Updates for Logistic Regression

In logistic regression we are trying to find the best betas such that we minimize the cross-entropy of the predicted probabilities with their true labels. Or in math:

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad -\sum_{i=0}^{n} y_i \, log\big(\sigma(\boldsymbol{X}_i\boldsymbol{\beta})\big) + (1-y_i)\, log\big(1-\sigma(\boldsymbol{X}_i\boldsymbol{\beta})\big).$$

For the remainder of this example we will focus on the situation with a single data point, as we'd see in SGD, $\boldsymbol{x}$ to simplify notation. This makes our loss function, with respect to one data point:

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad l(\boldsymbol{\beta}, \boldsymbol{x}, y) = -y\, log\big(\sigma(\boldsymbol{x}\boldsymbol{\beta})\big) - (1-y)\, log\big(1-\sigma(\boldsymbol{x}\boldsymbol{\beta})\big).$$

Our gradient descent problem, then, is to find the derivative of the loss function, l, with respect to the betas, $\frac{\partial l}{\partial \boldsymbol{\beta}}$ in order to minimize the loss by iteratively updating our betas via step two in the pseudocode above.

To do this we'll use the chain rule:

$$\frac{\partial l}{\partial \boldsymbol{\beta}} = \frac{\partial l}{\partial \sigma(u)} \frac{\partial \sigma(u)}{\partial u(\boldsymbol{\beta})} \frac{\partial u(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}},$$

where $u(\boldsymbol{\beta}) = \boldsymbol{x}\boldsymbol{\beta} \Rightarrow \frac{\partial u}{\partial \boldsymbol{\beta}} = \boldsymbol{x}$.

$$\frac{\partial l}{\partial \sigma} = -\frac{y}{\sigma} - \left(\frac{1-y}{1-\sigma}(-1)\right) = \frac{1-y}{1-\sigma} - \frac{y}{\sigma}$$

$$\begin{aligned}
\sigma(u) &= \frac{1}{1+e^{-u}} = \big(1+e^{-u}\big)^{-1} \\
&\Rightarrow \sigma + \sigma e^{-u} = 1 \\
&\Rightarrow \frac{1-\sigma}{\sigma} = e^{-u} \\
&\Rightarrow e^{-u} = \frac{1}{\sigma} - 1 \\
\Rightarrow \frac{\partial \sigma}{\partial u} &= -\big(1+e^{-u}\big)^{-2}\big(-e^{-u}\big) = \big(1+e^{-u}\big)^{-2}\big(e^{-u}\big) \\
&= \sigma^2(\frac{1}{\sigma} - 1) \\
&= \sigma(1-\sigma)
\end{aligned}$$

Putting all of this together.

$$\begin{aligned}
\frac{\partial l}{\partial \boldsymbol{\beta}} &= \left(\frac{1-y}{1-\sigma} - \frac{y}{\sigma}\right)\big(\sigma(1-\sigma)\big)\boldsymbol{x} \\
&= \big((1-y)\sigma - y(1-\sigma)\big)\boldsymbol{x} \\
&= \big(\sigma(\boldsymbol{x}\boldsymbol{\beta}) - y\big)\boldsymbol{x}
\end{aligned}$$