Neural Network Binary Classification Using Gradient Descent

Jasmine Khalil

1 Introduction

Single-class image classification can be efficiently done with gradient descent as an optimization algorithm to train a neural network using supervised learning. In this analysis, I walk through the mathematics behind logistic regression with gradient descent for neural networks.

2 Logistic Regression

Logistic regression is a learning algorithm for binary classification.

Therefore, a neural network with this model only has two possible outputs, 1 or 0, as the prediction of whether the object is present in the input image.

Given an input image x, we want our model to output a prediction. Where x is a feature vector consisting of the number of color channels x m x n entries. Parameters are w, which is the same size as the feature vector, and b, which is a real number. Using x, w, and b we derive our output as:

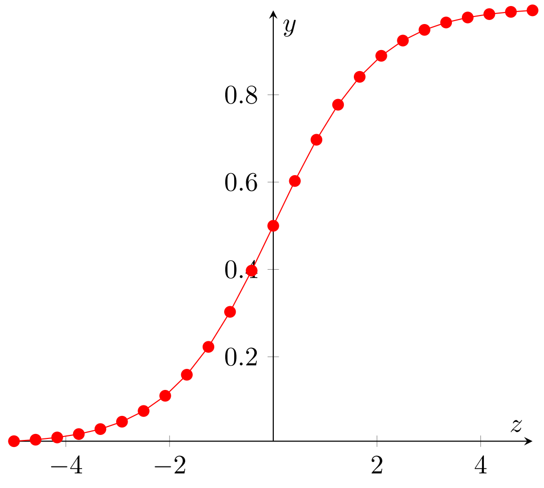
$$\hat{y} = \sigma(z) \tag{1}$$

$$z = w^{\mathsf{T}}x + b \tag{2}$$

The sigmoid function is applied to z because our output prediction must be between 0 and 1, and z alone could be any value much greater or lower than that. The sigmoid function constrains the output between 0 and 1:

$$\lim_{x\to\infty} \frac{1}{1+e^{-z}} = 1, \lim_{x\to-\infty} \frac{1}{1+e^{-z}} = 0 \tag{3}$$

Sigmoid Curve



2.1 Loss Function

In logistic regression, the loss function $l(\hat{y}, y)$ measures loss on a single training example and is given by:

$$l(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log 1 - \hat{y}) \tag{4}$$

If $y = 1$, we are trying to make $\hat{y}$ as large as possible (1):

$$l(\hat{y}, y) = -\log \hat{y} \tag{5}$$

If $y = 0$, we are trying to make $\hat{y}$ as small as possible (0):

$$l(\hat{y}, y) = -\log (1 - \hat{y}) \tag{6}$$

2.2 Cost Function

The cost function, $J(w, b)$, measures how well we are doing the model is performing on the entire training set. It is the cost of the parameters w and b, and is given by:

$$J(w, b) = -\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)} \log 1 - \hat{y}^{(i)}) \tag{7}$$

Where m is the number of training examples. Our goal is to find w and b to minimize the overall cost function.

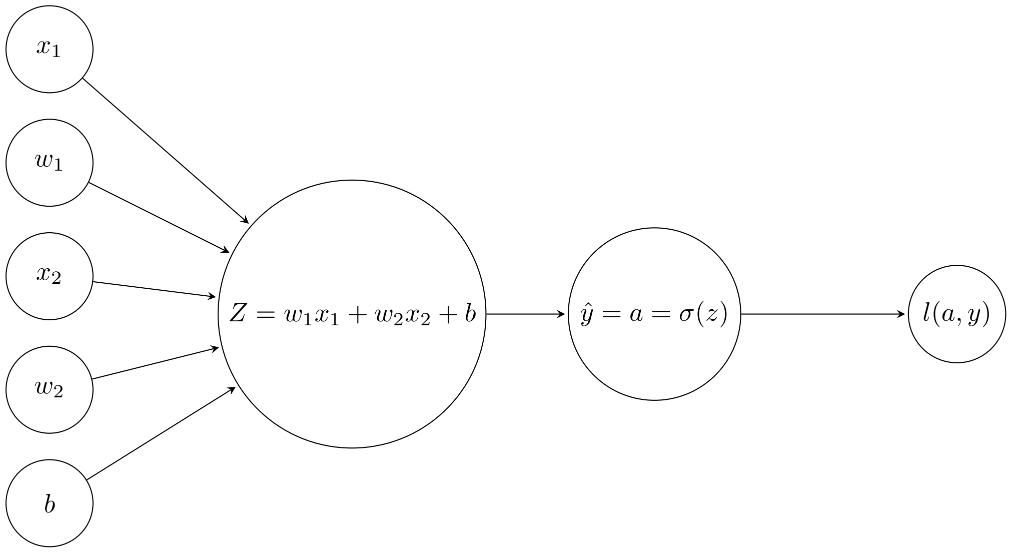
2.3 Gradient Descent

The gradient descent algorithm is used to find the parameters of the cost function to minimize it. The cost function is a convex function, so there are no multiple local optima. The first step is to initialize w and b to values like zero, but this is not vital because regardless of where we initialize, we will reach the same local optima. Gradient descent starts at the initialized point and moves toward the direction of the steepest descent. We update the parameters by:

$$w = w - \alpha \frac{dJ(w, b)}{dw} \tag{8}$$

$$b = b - \alpha \frac{dJ(w, b)}{db} \tag{9}$$

Where α is the learning rate that controls the step size in the direction of the steepest descent.



Using backpropagation to compute the derivatives, for i = 1 to m:

$$\frac{dl}{da} = \frac{-y}{a} + \frac{1-y}{1-a} \tag{10}$$

$$\frac{dl}{dz} = \frac{dl}{da}\frac{da}{dz} = (\frac{-y}{a} + \frac{1-y}{1-a})(a - a^2) \tag{11}$$

$$\frac{dl}{dz^{(i)}} = dz = a^{(i)} - y^{(i)} \tag{12}$$

$$\frac{\partial l}{\partial w_1} = x_1 * dz \tag{13}$$

$$\frac{\partial l}{\partial w_2} = x_2 * dz \tag{14}$$

$$\frac{\partial l}{\partial b} = dz \tag{15}$$

2.4 Vectorization

Stacking together all m inputs in different columns, we form a matrix X of shape (n_x, m):

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix} \tag{16}$$

Using vectorization, we can compute $z^{(1)}, z^{(2)}, ..., z^{(m)}$ in one step by constructing the row vector, Z:

$$Z = \begin{bmatrix} z^{(1)} & z^{(2)} & z^{(3)} & \dots & z^{(m)} \end{bmatrix} \tag{17}$$

$$dZ = \begin{bmatrix} dz^{(1)} & dz^{(2)} & dz^{(3)} & \dots & dz^{(m)} \end{bmatrix} \tag{18}$$

$$Z = w^T X + [b \quad b \quad \dots \quad b] \tag{19}$$

We can do the same for each prediction, $a^{(i)}$:

$$A = \begin{bmatrix} a^{(1)} & a^{(2)} & \dots & a^{(m)} \end{bmatrix} \tag{20}$$

$$A = \sigma(Z) \tag{21}$$

By also stacking the actual labels of each training example, we get Y:

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \dots & y^{(m)} \end{bmatrix} \tag{22}$$

From (12) and (17), we get:

$$dZ = A - Y = \begin{bmatrix} a^{(1)} - y^{(1)} & a^{(2)} - y^{(2)} & \dots & a^{(m)} - y^{(m)} \end{bmatrix} \tag{23}$$

Before vectorization, we updated derivatives w and b by:

$$\begin{aligned} dw &= 0 & db &= 0 \\ dw &+= x^{(1)} dz^{(1)} & db &+= dz^{(1)} \\ & & &\vdots \\ dw &+= x^{(m)} dz^{(m)} & db &+= dz^{(m)} \end{aligned}$$

After vectorizing, db and dw each become single operations:

$$db = \frac{1}{m}\sum_{i=1}^{m} dz^{(i)} \tag{24}$$

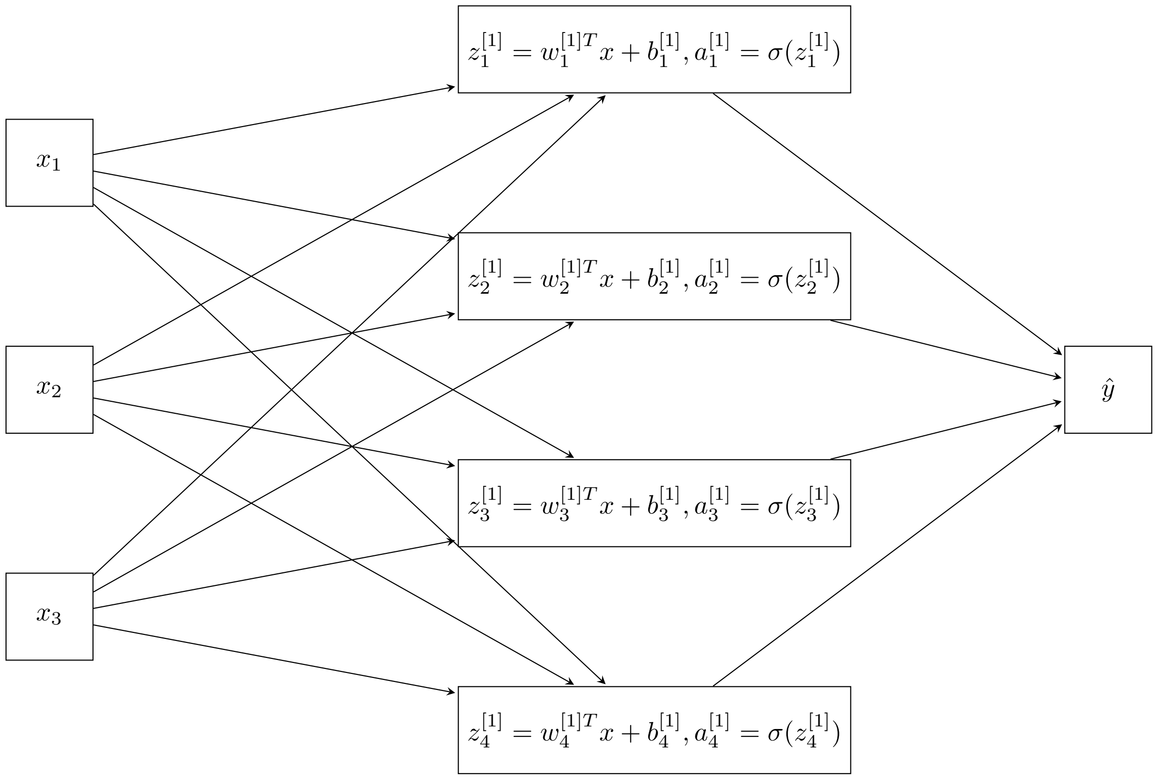
$$dw = \frac{1}{m} X \, dZ^T \tag{25}$$

Using the vectorized parameter derivatives, we can implement logistic regression by:

$$\begin{gathered}
Z = w^T X + b \\
A = \sigma(Z) \\
dZ = A - Y \\
dw = \frac{1}{m} X \, dZ^T \\
db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)} \\
w = w - \alpha \, dw \\
b = b - \alpha \, db
\end{gathered} \tag{26}$$

2.5 Logistic Regression as a Neural Network

Where l = layer number, and i = node number, logistic regression in a two-layer neural network would look like:


$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, a_1^{[1]} = \sigma(z_1^{[1]})$
x_1
$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, a_2^{[1]} = \sigma(z_2^{[1]})$
x_2
$\hat{y}$
$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, a_3^{[1]} = \sigma(z_3^{[1]})$
x_3
$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, a_4^{[1]} = \sigma(z_4^{[1]})$

Each logistic regression unit has a parameter vector, w, and by stacking each parameter, we get a $4x3$ matrix. Then, multiplying it by a matrix made of the input features, $a^{[0]}$, a $3x1$ matrix, and adding the result to $b^{[1]}$, a $4x1$ matrix, we have vectorized $z_i^{[l]}$ into a single matrix, $Z^{[1]}$.

$$Z^{[1]} = \begin{bmatrix} — & w_1^{[1]T} & — \\ — & w_2^{[1]T} & — \\ — & w_3^{[1]T} & — \\ — & w_4^{[1]T} & — \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]T} \\ b_2^{[1]T} \\ w_3^{[1]T} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T}x + b_1^{[1]T} \\ w_2^{[1]T}x + b_2^{[1]T} \\ w_3^{[1]T}x + b_3^{[1]T} \\ w_4^{[1]T}x + b_4^{[1]T} \end{bmatrix} \tag{27}$$

Vectorizing across a single training example is not useful; instead, we need to vectorize across multiple training examples. With (16), we vectorize $Z^{[1]}$, $A^{[1]}$, $Z^{[2]}$, and $A^{[2]}$ for multiple training examples.

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[2](2)} & \dots & z^{[m](m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[2](2)} & \dots & a^{[m](m)} \\ | & | & & | \end{bmatrix}$$

Where (i) is the training example number and $[v]$ corresponds to the different hidden units.

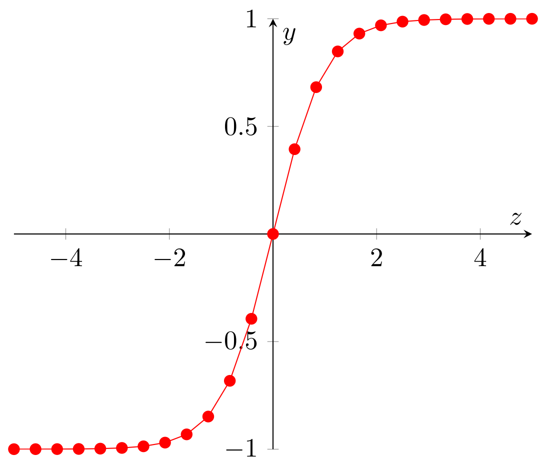
2.6 Activation Functions

In the forward propagation steps, we used the sigmoid function as the activation function. A commonly better-performing activation function is the hyperbolic tangent function, $a = tanh(z)$, which is:

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{28}$$

The hyperbolic tangent function is a shifted function of the sigmoid function, which goes from $(-1, 1)$.

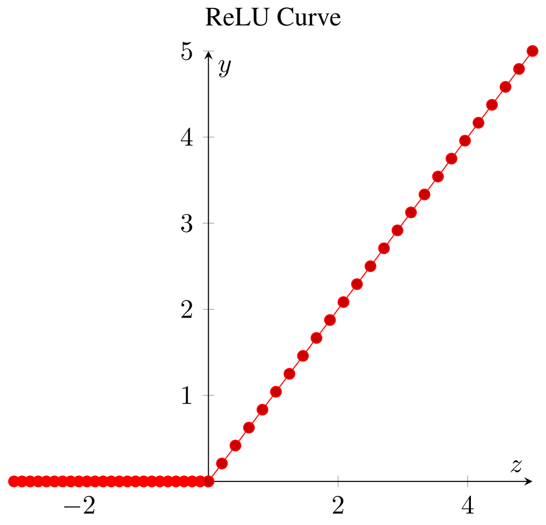
Hyperbolic Tangent Curve



$tanh(z)$ is typically a better activation function for hidden layers than the sigmoid function because the average of the activation is more centered, making learning for the next layer easier. An exception is in the output layer when using binary classification where having $0 \leq \hat{y} \leq 1$ is more useful than $-1 \leq \hat{y} \leq 1$.

One of the disadvantages of both $tanh(z)$ and $sigmoid(z)$ is as z approaches ∞ and $-\infty$, the derivatives of both curves approach 0, slowing down gradient descent. An activation function that tackles this issue is the Rectified Linear Unit (ReLU).

$$ReLU(z) = \tag{29}$$



When $z > 0$, the slope of the graph is 1 and 0 when $z < 0$, so the neural network will learn faster than when using other activation functions.

3 Neural Network

3.1 Gradient Descent Overview

The cost function of a binary classifier:

$$J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m}\sum_{i=1}^{m} l(\hat{y}, y) \tag{30}$$

For a binary classifier, the loss function would be (7), the same as for logistic regression. To start gradient descent, we initialize the parameters to random values, then:

- Compute $\hat{y}^{(i)}$ for $i = 1, ..., m$
- Compute the derivatives of the cost function parameters $dw^{[1]}$, $db^{[1]}$, $dw^{[2]}$, $db^{[2]}$ and update each parameter as in (8) and (9).

3.2 Forward Propagation

For a single training example, x, the forward propagation of a layer takes an input of $a^{[l-1]}$ from the previous layer and outputs $a^{[l]}$. The general forward propagation steps are:

$$\begin{gathered} z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]} \\ a^{[l]} = g^{[l]}(z^{[l]}) \end{gathered} \tag{31}$$

Where the function g is the activation function. For the entire training set, we vectorize the steps for each training example:

$$\begin{gathered} Z^{[l]} = w^{[l]}X + b^{[l]} \\ A^{[l]} = g^{[l]}(Z^{[l]}) \end{gathered} \tag{32}$$

$$X \xrightarrow{a^{[0]}} w^{[1]}, b^{[1]} \xrightarrow{a^{[1]}} w^{[2]}, b^{[2]} \xrightarrow{a^{[2]}} \cdots w^{[l]}, b^{[l]} \xrightarrow{a^{[l]}, \hat{y}}$$

3.3 Backward Propagation

In the backpropagation of layer l, the input would be $da^{[l]}$, and output $da^{[l-1]}$ The general steps for the backpropagation of a single training example are:

$$\begin{aligned} dz^{[l]} &= da^{[l]} * g^{[l]'}(z^{[l]}) \\ dw^{[l]} &= dz^{[l]} * a^{[l-1]}) \\ db^{[l]} &= dz^{[l]} \\ da^{[l-1]} &= w^{[l]T} * dz^{[l]} \end{aligned} \tag{33}$$

With vectorization, the steps to backpropagation become:

$$\begin{aligned} dZ^{[l]} &= dA^{[l]} * g^{[l]'}(Z^{[l]}) \\ dW^{[l]} &= \frac{1}{m} dZ^{[l]} * Z^{[l-1]}T) \\ db^{[l]} &= \frac{1}{m} \sum_{i=1}^{m} dZ^{[l]} \\ dA^{[l-1]} &= W^{[l]T} * dZ^{[l]} \end{aligned} \tag{34}$$

$$\xleftarrow{da^{[0]}} w^{[1]}, b^{[1]} \xleftarrow{a^{[1]}} w^{[2]}, b^{[2]}, dz^{[2]} \xleftarrow{a^{[2]}} \cdots \xleftarrow{da^{[l-1]}} w^{[l]}, b^{[l]}, dz^{[l]} \xleftarrow{da^{[l]}}$$

$da^{[0]}$ is a derivative of the input features, which serves no purpose, so it can be disregarded. Using the final derivative terms, we update the parameters as in (8) and (9):

$$w = w - \alpha \, dw^{[l]} \tag{35}$$

$$b = b - \alpha \, db^{[l]} \tag{36}$$

4 Conclusion

In this analysis, I discuss the transition from the loss to cost function and the breakdown of backward propagation and forward propagation steps for both single and multiple input features. Through vectorization, particularly in the backward propagation steps, unnecessary for-loops are eliminated, and by utilizing the optimum activation function, we also improve the learning rate of our model. Starting with a single input and then progressing to m input features, I have broken down logistic regression as a simple model to perform binary classification.