Introduction to Stochastic Gradient Descent

Explained through Neural Networks

Landon Buell

27 January 2020

1 Introduction

Gradient Descent algorithms lie at the heart of several machine learning algorithms [3]. In general, the goal of the algorithm is find a set of parameters that minimize the value of a particular function, which we call the *Cost Function* or *Objective Function* [1]. To do this, we apply a procedure from multivariate calculus that repeats a procedure until a local minimum of that particular function is found. Note that while we ideally want to find a global minimum (the minimum of the whole function), this is computationally unrealistic, so local minima of the function are used in it's place.

Suppose we have some scalar defined function, f of n independent variables. We notate this as:

$$f = f(x_0, x_1, ..., x_{n-2}, x_{n-1}) \tag{1}$$

The gradient of the function, returns a vector, with each element given by the partial derivative of f with respect to the i-th variable:

$$\nabla\big[f(x_0, x_1, ..., x_{n-2}, x_{n-1})\big] = \Big[\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_{n_2}}, \frac{\partial f}{\partial x_{n-1}}\Big] \tag{2}$$

Geometrically, the gradient provides a vector that points in the direction that causes the value of f to increase the fastest. In a two dimensional function, where the output is the 3rd dimension, this gives the direction of steepest ascent. Note that if the function of of n variables, then this operation take place in n-dimensional vector space.

The general procedure of the Gradient Descent Algorithm is:

1. Pick a 'starting point', p_0 in the n-d space. This is done by evaluating the function f for each of it's n input variables.
2. Compute the gradient of f, eq. (2) at the point p_0. Multiply the gradient by -1. This is the direction of steepest *descent*.

3. Follow the negative gradient to reach a new point p_1.
4. Repeat steps 2 and 3 until the value of $-\nabla f$ returns 0. This indicates that a local minimum, or a saddle point has been found.

Some gradient descent algorithms may repeat this whole procedure with a series of initial points, each time tracking the minimum and corresponding parameters. This way, each time the algorithm is repeated, a new minimum is attained, which allows for a greater chance to find a successively lower value. While it seems appealing at first thought, it is work noting that this whole algorithm as outlined is *very* computationally expensive.

2 Considerations and Conventions

2.1 Structure

It is important to understand the architecture of a convolution neural network (CNN) at a very basic level. In it's simplest form as well will describe, a neural network is a collection of layers of functions (often called *nodes* or *neurons*) [3]. The exact amount of layers in a network depends on and the exact number of neurons in each layer depends on the exact type of task that the network seeks to accomplish. Currently, there is no formal rule to outline this exactly.

The entry point of a neural network, called *Layer 0* is the set of functions (or values) that receives an initial piece of data. The number of these inputs neurons in this layer corresponds to the number of *features* in the base data set. Thus if a sample of data has n_0 features, then there are n_0 neurons in this layer 0 of the neural network [2].

To move to the next layer, an operation is applied to all of the values in the previous layer. We model this with standard matrix multiplication. notation conventions for this will outlined in the next section. After each matrix multiplication is applied, the input is effectively transformed into the next layer of the network. It is important to know that although layer 0 has n_0 neurons, any other layer may have a different number of neurons in it, This difference is handled by the dimension of the matrix that allows for the transformation between adjacent layers.

In a network with L layers, there are $L-2$ matrix multiplications required (due to 0 - indexing). Once the $L-2$ matrix has been applied to layer $L-2$, the resultant layer, $L-1$, is effectively the output of the network. In the case of a CNN being used for a classification problem, the number of nodes in the final layer of the network, corresponds to the number out target classes. Thus if we built a *K-Folds* classifier, the output layer of the CNN would have $n_{(l-1)} = K$ nodes.

2.2 Notation

The mathematical description of stochastic gradient descent (SGD) classifiers is largely based in linear algebra [3]. This means a great deal of matrix vector indexing is required to fully describe the procedure. For this work, I will be using a standard of 0 - indexing all objects. I outline a notional convention for this work:

- A scalar:
$$n_l \tag{3}$$
is the number of neurons (also called nodes or functions) in the l-th layer.

- A vector:
$$\vec{x}_i^l \tag{4}$$
is the i-th entry in the vector x in the l-th layer of a network. The vector that describes layer l of the network has n_l rows and 1 column. Thus the vector $\vec{x}^1$ has a n_l number of components in it.

- A matrix:
$$W_{i,j}^l \tag{5}$$
is the entry in the i-th row, and the j-th column of matrix W that operates on the l-th layer of a network. The matrix that operates on layer l has $n_{(l+1)}$ rows and n_l columns

Mathematically, each neuron, contains a number that can be operated on.

3 Convolution Neural Network Procedure

Suppose we have a linear neural network, with L layers of neurons, each containing n_l number of neurons in it. The first layer of neurons is called the *input layer*, which contains n_0 neurons. This corresponds to the number of input features given to the network. Each input feature then fills in the entry of it's respective neuron. Thus, after presenting the network with an array of raw data, the x^0 vector then becomes:

$$\vec{x}^0 = \left[x_0^0, x_1^0, x_2^0, ..., x_{n_0-2}^0, x_{n_0-1}^0\right]^T \tag{6}$$

Understand that every entry in this vector is a numerical quantity. In the case of a computer, they are generally floating-point numbers, often normalized between 0 and 1 to ease computations.

The network must then feed this information , in the form of a column vector to the next layer of itself. It must take the vector $\vec{x}^0$ and transform it to into the vector $\vec{x}^1$ containing the entries in the neurons in 1-th layer. Recall that these layers may have a different number of neurons. We perform this transformation in the form of a matrix of weights, which we denote as W^0.

The weighting matrix W^0 operates on vector $\vec{x}^0$ to produce the entries in the vector $\vec{x}^1$. More generally, matrix W^l operates on layer/vector x^l to produce layer/vector x^{l+1}. The dimensions of matrix W^l must then be n_{l+1} rows by n_l columns. In a network with L layers, there are L vectors and $L-1$ weighting matrices.

The process of matrix multiplication makes up the bulk of the *feed-forward* algorithm for neural networks. We can generalize this reoccurring procedure by the matrix vector equation below.

$$x^{l+1} = W^l x^l \tag{7}$$

Where the superscript l indicates an arbitrary integer layer number. This process repeats for $L-1$ matrix multiplications, often making a network a very computationally expensive operation. Once the vector x^{L-1} is produced, the network has finished it's algorithm and a result is then the final output of the network. Recall that this layer is called the *output layer* and has n_{L-1} number of neurons in it.

4 The Weighting Matrices

The actual procedure of this type of neural network is essentially this basic recursive linear algebra problem. It then raises the questions, how is each matrix, W^l built? What are it's elements and what do they do? If we examine the matrix-vector equation (7) we can break this down further into it's constituent parts. For simplification of super scrips and subscripts, let the *next* layer, $l+1$ be the vector y with a entries, and the *previous* layer, l be the vector x with b entries. Thus the matrix W^l has a rows, and b columns. each entry is denoted as $W_{i,j}$.

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{a-2} \\ y_{a-1} \end{bmatrix} = \begin{bmatrix} W_{0,0} & W_{0,1} & \dots & W_{0,b-2} & W_{0,b-1} \\ W_{1,0} & W_{1,1} & \dots & W_{1,b-2} & W_{1,b-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ W_{a-2,0} & W_{a-2,1} & \dots & W_{a-2,b-2} & W_{a-2,b-1} \\ W_{a-1,0} & W_{a-1,1} & \dots & W_{a-1,b-2} & W_{a-1,b-1} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{b-2} \\ x_{b-1} \end{bmatrix} \tag{8}$$

Keep in mind that a may not be equal to b. We can now use this to understand a little bit more specifically what the weighting matrix is doing, and what each entry in the operation does.

From linear algebra, the entry y_i in the column vector y is given by the dot product between the $W_i,:$ row and the column vector x. We can denote this as:

$$y_i = \sum_{j=0}^{b-1} x_j W_{i,j} = (W_{i,0}x_0) + (W_{i,1}x_1) + ... + (W_{i,b-2}x_{b-2}) + (W_{i,b-1}x_{b-1}) \tag{9}$$

References

[1] James, Gareth, et al. An Introduction to Statistical Learning with Applications in R. Springer, 2017.

[2] Géron Aurélien. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly, 2017.

[3] Goodfellow, Ian, et al.*Deep Learning*. MIT Press, 2017.

[4] Petrik, Marek. "Introduction to Machine Learning." Machine Learning. 22 Jan. 2020, Durham, New Hampshire.