Name(s):

Challenge: Create a New Regression Algorithm (ML Club 0608)

Gradient Descent in Practice

TA: Anish Lakkapragada

In this challenge you will learn gradient descent in a simple linear regression case, and then you will be able to create your own regression algorithm. Just submit this assignment as your PhD thesis!

1 Gradient Descent

1.1 Construction

Let's define our variables now to make things as simple as posible. We are given a dataset X, which for our purposes will contain only one variable, and a set of labels y (i.e. correct predictions). Our model is given by a function $f(x)$ that has an internal set of parameters θ. Our job is to adjust θ in a way where the outputs of $f(x)$ from inputs X are closest to y as possible.

1.2 Linear Regression Example

We have thus far laid the groundwork for what any generic kind of model could be. Let us simplify this to the linear regression case, where we define

$$\hat{y}_i = f(x_i) = \theta_0 x_i + \theta_1$$

where $\hat{y}$ represents our predictions and x_i represents the ith prediction from our train set X. In this case, θ is the set of parameters θ_0, θ_1. Our job is to change θ_k, where k refers to any parameter in the set of θ, in a way that will make $\hat{y}$ match more closely with y.

We can't manage what we can't measure - we should quantify how well the predictions of $f(x)$ as given by a specific set of θ are. Or we could we do the opposite of measuring how bad the model is given the current parameters.

$$J(\theta) = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 = \sum_{i=1}^{N} (f(x_i, \theta) - y_i)^2$$

Here, J is a function that takes θ as input and returns the sum of the squared distances between the predictions $\hat{y}_i$ and the label y_i. We want to minimize this as much as possible - this is our objective or goal, thus why this is denoted as an objective function J. This specific function of J is known as the squared sum of residuals (SSE.) Note that the function f is exactly the same before, except here we have written it taking in θ as another parameter. This is to introduce the idea that θ is constantly changing to minimize $J(\theta)$ in the algorithm Gradient Descent, which we will introduce in the next section.

1.3 Gradient Descent Algorithm

Gradient Descent is a very fundamental algorithm to basically all of machine learning. A big subset of machine learning, known as *supervised learning*, can be defined as having some function $f(x, \theta)$ and needing to change θ in a way that minimizes $J(\theta)$. For all we know, $f(x, \theta)$ can range from a humble regression model to a complex neural network or beyond.

Enough description, let's get on with it! In gradient descent, the model first takes a guess at what the values of the parameter set θ could be. Practically speaking, this means that θ_k for our linear regression algorithm will be set to random scalars.

Gradient Descent is an iterative process, where at each iteration we try to find some change $\Delta\theta_k$ for every parameter to add to the current value of θ_k in a way that will minimize J.

The way we figure out $\Delta\theta_k$ is by seeing the derivative of J at the current point of θ_k. If you look at a parabolic curve, you'll see that in the set of x where the curve is increasing (derivative > 0) if you move in the direction of the derivative you will ascend. Similarly, where the derivative is negative (curve decreasing), if you move in the direction of this derivative (negative) you will ascend on the curve. Moving in the direction of just means adding the derivative of the curve at the current x value to the current x value itself, and then checking the new value of the functions.

In short, the derivative gives the direction of steepest ascent to rise on any curve. If we wanted to maximize J, we should just set $\Delta\theta_k$ to the derivative $\frac{dJ(\theta)}{d\theta_k}$. However, we want to minimize J so we set $\Delta\theta_k$ to the negative of $\frac{dJ(\theta)}{d\theta_k}$ to move the opposite way. This is where the name comes from - gradient descent stands for descent by gradient (derivative in many dimensions). We calculate this change in the current parameters for every iteration until the change reaches 0, indicating convergence has been met.

$$\theta_t = \theta_{t-1} + \Delta\theta_{t-1} = \theta_{t-1} - \frac{dJ(\theta_{t-1})}{d\theta_{t-1}}$$

This details gradient descent moving from iteration $t-1$ to t. Note that I have dropped the k in θ_k for clarity; the θ above could represent either θ_0 or θ_1 in our linear regression function f.

Congratulations, you have just gotten reached a solid understanding of Gradient Descent [1]!

1.4 Gradient Descent for Linear Regression From Scratch

Understanding gradient descent is a huge accomplishment. Kudos to you for making it this far! However, to understand it deeper and eventually create your *own* regression algorithms - it is best to try to go even deeper into gradient descent and calculates the derivatives themselves.

We will do just that for the aforementioned linear regression function $f(x_i) = \theta_0 x_i + \theta_1$ where we try to minimize the objective SSE function. To do this, we just need to calculate $\frac{dJ(\theta)}{dJ(\theta_0)}$ and $\frac{dJ(\theta)}{dJ(\theta_1)}$. Let's start with the first one.

$$\frac{dJ(\theta)}{d\theta_0} = \sum_{i=1}^{N} 2(\hat{y}_i - y_i)\frac{d\hat{y}_i}{d\theta_0}$$

Okay, so know we need to just calculate $\frac{d\hat{y}_i}{d\theta_0}$. Since θ_1 is just a constant in yielding $\hat{y}$, all it really is $\frac{d\theta_0 x_i}{d\theta_0}$ or x_i. Thus we get $\frac{dJ(\theta)}{d\theta_0} = \sum_{i=1}^{N} 2(\hat{y}_i - y_i)x_i$. Solving for $\frac{dJ(\theta)}{d\theta_1}$ gives basically the same as above with 1 instead of x_i.

1.5 Recap

You may be wondering why to calculate the derivatives on hand as we just did. It's because it helps understand and design models in more detail. For example, what if we choose J to take the absolute value, not square, of $f(x_i, \theta) - y_i$?

If you calculate the derivatives by hand, you will see that this will lead to quite a lot of problems as you cannot differentiate the absolute value function. This can also be extended to the fact that if our model $f(x_i)$ is not differentiable, our regression model will work.

[1] Halloween is coming up - if you want to get even more scared about gradient descent feel free to read this article which shows that gradient descent is derived from a first-degree (meaning it only takes the first-order derivative/gradient) Taylor Series of the objective function J: https://www.cs.princeton.edu/courses/archive/fall18/cos597G/lecnotes/lecture3.pdf.

If you are further interested in this kind of optimization, perhaps look into *closed-form optimization*. This is where instead of iteratively changing the parameters as Gradient Descent does, you solve for the values of θ by solving for what values of θ make $\frac{dJ(\theta)}{d\theta_k}$ equal to 0.

The main point to remember that the above process of gradient descent is applicable for any function. We can set $f(x_i)$ to any function we want - from logistic regression to neural networks - as long as it is differentiable.

2 Creating a Regression Algorithm

2.1 Step 1: Create a Model

The first step of creating a novel machine learning algorithm is to decide what your prediction function will be. This is more complex than it seems - you have to think about the type of data you want to model.

For example, if you want to create an exponential regression model you could model your data as $f(x_i) = \alpha e^{\beta x_i}$, where α, β are the parameters to be adjusted from Gradient Descent. If you are stuck for an algorithm to create, think about all the types of data curves out there (cyclic, exponential,etc.). For example, you have sinusoidal (models fluctuating curve), polynomial, exponential, etc. curves and shapes. Have a try at it below!

Prediction Function for Model $f(x_i) =$
Parameters for Gradient Descent:

2.2 Step 2: Choose an Objective Function

As part of gradient descent, there needs to be an objective function for your model to reduce. You can stick with SSE, or you can try to create something new (e.g. the difference between predictions and labels could be numerically far away in exponential regression, but in context be useful.) Feel free to just choose SSE if that is easier / more appropriate. Remember that the θ denoted below represents the set of all of the parameters labeled above in step 1.

Objective function $J(\theta)$:

2.3 Step 3: Solve Gradient Descent

With the above objective function J to minimize for your model defined by function f, it is finally time to do the last step: calculate the derivatives that will be required by gradient descent. Stated more clearly, this means that you will need to calculate the derivative of your objective function with respect to each of the parameters labeled in step 1. This will test whether everything - the decided prediction function f and objective function J will work together.

Parameter 1 Derivative:
Parameter 2 (if required) Derivative:
Add more as necessary.

2.4 Step 4: Congratulate Yourself

Congrats!!! You have just created a new algorithm! This is a big accomplishment.