Gradient Descent Process

Gradient descent is an algorithm that allows you to fit any kind of model to a set of points with any kind of measure of error. This document will walk you through the basic steps of how to do it. As a motivating example, we will refer to the function $y = ax^2 + bx + c$, but know that you will need to change the computations for different types of functions you consider.

Outline

1. **Graph the data and assess potential types of functions** that could fit your dataset.
    a. Here, we will assume that your visual inspection is a U shape, so you choose a degree 2 polynomial.
2. **Split the data into training, validation, and testing sets**
    a. Typically, your validation and testing sets are as much as 20% of your data and as little as 5%.
    b. In the example, we first remove the testing set, then the validation set. This is because you will use both the training and validation to construct your final model

```
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.1, random_state=0)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.1,random_state=0)
```

3. **Select your error function**
    a. Mean squared error (MSE) and mean absolute error (MAE) are typical. Mean absolute error is favored in this class.
    b. The function for MAE is

$$E = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

4. **Compute the partial derivatives** of your error function with respect to the parameters of your model:
    a. In our model, we will look at the partial derivatives of E with respect to a, b, c.
    b. For example, for MAE with our model, the derivative with respect to a is:

$$E_a = \frac{1}{n}\sum_{i=1}^{n} sign\left(y_i - (ax^2 + bx + c)\right) \cdot x^2$$

5. **Pick initial values for your parameters** (coefficient)
    a. This means you pick an initial value for a, b, c

    b. A great way to do this is plot your guesses on top of your data play around with values until you get something close-ish to your data.
6. **Pick initial values for your hyperparameters**.
    a. **Epochs** – these are the number of iterations (times in the loop you will do). 1000 is usually more than enough.
    b. **Learning rate** – this is what you will multiply your derivatives by. If you are unsure, a small learning rate, like 0.001, is a good place to start.
7. **Perform gradient descent on your training set; evaluate with your validation set.**
    a. Compute predictions. (ypred) with the current value of the parameters
    b. Compute the derivatives with the current value of the parameters
    c. Update the parameters using the gradient descent formula
    d. Compute the loss/error of the training and the validation sets
8. **Repeat this while tweaking your hyperparameters until you see convergence**.
    a. You want to see your errors get small and level out.
    b. You also don't want to see your training errors to be much lower than your validation errors.
    c. Things you can tweak in this process:
        i. The learning rate
        ii. Initial parameters
    d. You may decide to give up on the model and try a new one. If your errors are just consistently bad, it might be best to give up.
        i. Other things to try:
            1. Scale your data to be between -1 and 1.
            2. Try a different measure of error.
9. **Plot the loss for the validation and training set over the number of iteration/epochs.**
    a. This will help you assess how many epochs you should have/
10. Based on your previous work, **select your hyperparameters**.
    a. Number of epochs shouldn't be as high as what you did previously. You want a number that happens right when the errors level off but you don't want training to be significantly. Better than validation.
    b. The learning rate should be small enough to ensure a smooth convergence.
    c. Your initial parameters should allow you to be able to find good parameters in the end.
11. **Perform gradient descent with your selected hyperparameters using your training and validation set combined**.
12. Plot the actual data points and the model's prediction.
13. **Evaluate the loss using your testing function.** This will tell you how well your model performs.