It is of great interest to me to understand how the Adagrad algorithm works in relation to adapting the learning rate to the parameters which suites it better to work with large scale neural nets. The mathematical derivations on how to arrive at these said algorithms is key to provide with a great basis to understanding their performance and how to incorporate them into a real-world scenario.

To obtain certain minima point, we need to identify which ways to go as well as steps to take, and this is where gradient descent comes into play. Having seen and done some calculus on how to calculate any derivatives of given function, we see similarly how this mathematical notation of finding derivatives is also used in gradient descent algorithm to help in making the above decisions. It remains of interest to learn how minimization works for the cost function using the partial derivatives.

Generally, derivatives are mostly used in optimization algorithms like the gradient descent. There are several ways in which to obtain the derivative of any function as described within the blog post such as how to use power rule. This mathematical notation and procedure to find derivative of a variable raised to power have been broadly used in our previous classes.

The key takes away from the blog post is the emphasis on how different gradient descent optimization algorithms works and how each algorithm is suited to a particular task for better improved performance. It is worth noticing how we can use different gradient descent variants based on the amount of data to compute any gradient of the given objective function. Visualization of these algorithms is pivotal as it provides a clear understanding on their behaviour at each specific point within the slope something which is highlighted in these blog posts. In addition, which optimizer to use based on the input data is key to achieving the best results