Instructions: Make sure your questions and answers are on different pages. Do not include your name or any other identifying information. I will know that information from Canvas.

Question 1: What are the time complexities of the two mentioned methods of optimization, gradient descent and Newton's method, given a sample set of data with size *n* and individual gradient computations having *O(d)* time complexity? Based on their formulas, what are the main advantages of using either optimization method?

Question 2: In what way does the gradient descent method optimize deep learning algorithms?

Question 3: How does gradient descent differ between standard application and application to a strongly convex function?

Answer Question 1: For gradient descent, the time complexity can be calculated as follows: since each gradient calculation for each call of $f_i(w_t)$ has a time complexity of $O(d)$, there are n total calls made this way for each $i = 0, 1, \ldots n$. With the rest of the gradient descent formula being constant, the overall time complexity is $O(n*d)$.

For Newton's method, the time complexity can be calculated as follows: assume the same time complexity for calculating the gradients for each $f_i(w_t)$ call as in the gradient descent case. For Newton's method, according to http://en.citizendium.org/wiki/Newton%27s_method#Computational_complexity, the time complexity is $O(\log(n)*d)$, with the $\log(n)$ coming from the faster convergence of Newton's method as a result of using the Hessian of f.

Comparing these two time complexities, Newton's method is superior to gradient descent for finding the required minimum for optimization, as its time complexity is considerably smaller than that of gradient descent. However, Newton's method depends on calculating the second derivative of f, which could be infeasible for complicated instances of f. Thus, gradient descent is a better application when the Hessian of f requires more computational time to calculate than just its gradient, since gradient descent only makes use of the first-order derivative found in f's gradient.

Answer Question 2: The cumulative sum of each iterative application of gradient descent to the loss function f eventually becomes the global minimum of the loss function f^*. Through this observation, it can be noted that "the smallest gradient… after T iterations is getting smaller proportional to 1/T", from which it can be extrapolated that gradient descent eventually converges upon the optimized minimum.

Answer Question 3: In the standard application of gradient descent, the method eventually converges upon the optimum with a linear relationship to the learning rate. In the below equation for gradient descent, α_t signifies the learning rate, and the optimum weights are dependent upon the product of the gradient and learning rate.

$$w_{t+1} = w_t - \alpha_t \cdot \nabla f(w_t)$$

In the case of applying gradient descent to a strongly convex loss function, the following formula represents this specific relation: (μ is a parameter associated with convexity, T represents the number of iterations).

$$f(w_T) - f^* \leq (1 - \alpha\mu)^T \left(f(w_0) - f^*\right)$$

Here, it can be seen that, for a constant learning rate, the gradient converges to the optimum "exponentially quickly", as in the gradient descent has an exponential relationship to the learning rate. Thus, applying gradient descent to a strongly convex function results in much quicker optimization than normal gradient descent applications.