From Part 1, we have that $e_i = f(x_i; w) - y_i$ and $L_2 = \sum_i e_i^2 = \sum_i (f(x_i; w) - y_i)^2$.

To find the optimum, find the gradient of the loss function with respect to the parameters:
$\nabla_w L_2 = \frac{\partial}{\partial w}\left(\sum_i (f(x_i; w) - y_i)^2\right) = 2\sum_i \frac{\partial}{\partial w} f(x_i; w)(f(x_i; w) - y_i) = 2\sum_i f_w(x_i; w) e_i$

Considering the linear model we wish to develop, namely $f(x; w) = w_0 + \langle w, x\rangle$, the gradient should be understood as a vector in the n+1 dimensional parameter space, owing to special treatment of the "featureless" weight. Specializing the above formula, we obtain a set of expressions for the components of the gradient vector.
$\text{components}(\nabla_w L_2) = \left\{D_{w_j} L_2\right\}_{j=0}^{n} = \left\{D_{w_0} L_2\right\} \cup \left\{D_{w_j} L_2\right\}_{j=1}^{n} = \left\{2\sum_i e_i\right\} \cup \left\{2\sum_i x_{j,i}\, e_i\right\}_{j=1}^{n}$

Breaking this down more digestibly, the gradient consists of a sum of errors for the constant term and a sum of the corresponding feature times error for the feature weights. Reducing clutter, these can be simplified to:

Constant: $2\langle \mathbf{1}, e\rangle$

Feature j: $2\langle x_j, e\rangle$

Note that the errors are (an unknowable) function of the example data and the weights. Theoretically, we would achieve the minimum loss by setting each component of this gradient equal to zero and solving for the w vector. Unfortunately, as mentioned, the form of the function linking the examples and weights to the errors is unknowable to us, and even if it were knowable, it would most likely be impractical to actually solve the resulting system of equations.

With this in mind, our goal is to numerically solve for the parameters using a set of labeled examples forming training data. We then use what is derived above to teach the parameters about what values they should take. Since the loss is a convex function, the gradient will direct us through parameter space such that the fastest increase in loss is achieved. If we take some multiple of the negative gradient, we will instead be moving on the quickest path to the minimum at each step. A scalar is used to provide us with control over the algorithm stability and convergence time.

The update equation at step k takes the form: $w_k = w_{k-1} - r \cdot \nabla_w L_2(w_{k-1})$, where r is the learning rate. Different methods may allow us to programmatically update the learning rate as well, a sort of auto-tuning parameter. One article I found specifically mentions this strategy https://towardsdatascience.com/gradient-descent-algorithms-and-adaptive-learning-rate-adjustment-methods-79c701b086be which discusses adaptive learning rate methods.

One such strategy is the Nesterov accelerated gradient method, which essentially provides a two-step update. Choosing a value h, to multiply the historical learning rate, the update equations become
$$\begin{aligned} u_k &= h u_{k-1} + r \cdot \nabla_w L_2(w_{k-1} - h u_{k-1}) \\ w_k &= w_{k-1} - u_k \end{aligned}.$$