Optimisation Methods

Eavan Thomas Pattie

TASK 1

Implementation

Implementation of gradient descent with backtracking with torch starts by computing the gradients through a specified loss function and then computes the η for the current iteration by the rule given in the assignment. This is done in the `torch.no_grad()` context manager to avoid building up a computation graph. This saves us from having to detach variables later.

Instead of returning the optimum value the gradient descent with backtracking procedure is implemented as a generator. This is so that the full sequence of intermediate values and gradients can be analysed and used in other computations. This is seen in the `optim` method, which reads from the generator until the norm of the gradient falls below some given limit and then prints the value that gradient descent with backtracking had reached at that point in addition to some other statistics.

Difference in Rate of Convergence

In part 1.2 there are two similar functions that require a vastly differing number of iterations to converge. This can be explained by looking at the condition number of the two functions. The condition number is the ratio of the largest and smallest eigenvalues in the hessian of the function.

For the first function we can derive an expression for the condition number as follows.

$$f(x,y) = x^2 + 2y^2$$

$$H_f = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}$$

$$\kappa = 2$$

However, for the second function we see that the condition number is much larger

$$f(x,y) = x^2 + \frac{y^2}{100}$$

$$H_f = \begin{bmatrix} 2 & 0 \\ 0 & 0.02 \end{bmatrix}$$

$$\kappa = 100$$

As we can see the condition number for both functions is a constant and the condition number for the second function is significantly larger than the condition number for the first function. This explains why the second function is much harder to optimise and why it takes 201 iterations before it converges compared to the 2 iterations that the first function requires before it reaches convergence.

TASK 2

Implementation

The implementation of the gradient descent in part two follows the same principles as the implementation of gradient descent with backtracking from the previous task. The implementation of the cost function makes use of vectorised operations in the torch package. However, a straight forward implementation leads to nans in some cases. This is because torch at some point attempts to calculate $\log 0$. This can be fixed by adding some small value ε to the log.