Homework 4

Submission: This homework is not to be handed in for a mark. However, it is not optional, in the sense that you are still responsible for the content. We will release solutions, but you will get more out of it if you attempt the problems yourself first.

1. **Gradient descent.** We can get quite a bit of insight into the behavior of gradient descent by looking at how it behaves on quadratic functions. Suppose we are trying to optimize a quadratic function

$$\mathcal{C}(\boldsymbol{\theta}) = \frac{a_1}{2}(\theta_1 - r_1)^2 + \cdots + \frac{a_N}{2}(\theta_N - r_N)^2,$$

with each $a_i > 0$. We can exactly solve for the dynamics of gradient descent. In other words, we can find an exact formula for $\theta_i^{(t)}$, where t is the number of gradient descent updates.

(a) Derive the gradient descent update rule for each θ_i with learning rate α. It should have the form

$$\theta_i^{(t+1)} = \cdots,$$

where the right-hand side is some function of the previous value $\theta_i^{(t)}$, as well as r_i, a_i, and α. (It's an interesting and useful fact that the different θ_i's evolve independently, so we can analyze a single coordinate at a time.)

(b) Now let's introduce the *error* $e_i^{(t)} = \theta_i^{(t)} - r_i$. Take your update rule from the previous part, and write a recurrence for the errors. It should have the form

$$e_i^{(t+1)} = \cdots,$$

where the right-hand side is some function of $e_i^{(t)}$, r_i, a_i, and α.

(c) Solve this recurrence to obtain an explicit formula for $e_i^{(t)}$ in terms of the initial error $e_i^{(0)}$. For what values of α is the procedure stable (the errors decay over time), and for what values is it unstable (the errors grow over time)? You should find that large learning rates are more unstable than small ones, and that high curvature dimensions are more unstable than low curvature ones.

(d) Using your answer for the previous part, write an explicit formula for the cost $\mathcal{C}(\boldsymbol{\theta}^{(t)})$ as a function of the initial values $\boldsymbol{\theta}^{(0)}$. (You can write it as a summation over indices; you don't need to vectorize it.) As $t \to \infty$, which term comes to dominate? You'll find that the asymptotic behavior for optimal α roughly depends on the *condition number*

$$\kappa = \frac{\max_i a_i}{\min_i a_i}.$$

This supports the claim that narrow ravines are problematic for gradient descent.

(e) [**Optional, and advanced**] This part is optional, but you may find it interesting. We'll make use of eigendecompositions of symmetric matrices; see MIT OpenCourseware for a refresher:

`https://ocw.mit.edu/courses/mathematics/18-06sc-linear-algebra-fall-2011/positive-definite-matrices-and-applications/symmetric-matrices-and-positive-definiteness/`

It turns out we've actually just analyzed the fully general quadratic case. I.e., suppose we try to minimize a cost function of the form

$$\mathcal{C}(\boldsymbol{\theta}) = \frac{1}{2}(\boldsymbol{\theta} - \mathbf{r})^T \mathbf{A}(\boldsymbol{\theta} - \mathbf{r}),$$

where $\mathbf{A}$ is a symmetric positive definite matrix, i.e. a symmetric matrix with all positive eigenvalues. (This is the general form for a quadratic function which curves upwards.) Determine the gradient descent update for $\boldsymbol{\theta}$ in vectorized form. Then write a recurrence for the error vector $\mathbf{e} = \boldsymbol{\theta} - \mathbf{r}$, similarly to Part (1c). It will have the form

$$\mathbf{e}^{(t+1)} = \mathbf{B}\mathbf{e}^{(t)},$$

where $\mathbf{B}$ is a symmetric matrix. Determine the eigenvectors and eigenvalues of $\mathbf{B}$ in terms of the eigenvectors and eigenvalues of $\mathbf{A}$, and use this to find an explicit form for $\mathbf{e}^{(t)}$ and for $\mathcal{C}(\boldsymbol{\theta}^{(t)})$ in terms of $\boldsymbol{\theta}^{(0)}$. The result will be closely related to your answer from Part (1d).

2. In preparation for the second programming assignment, please read through the Autograd tutorial:

   `https://github.com/HIPS/autograd/blob/master/docs/tutorial.md`

   and the code for the multilayer perceptron example:

   `https://github.com/HIPS/autograd/blob/master/examples/neural_net.py`