Introduction to Machine Learning
Homework 5: Gradient Calculations and Nonlinear Optimization Solutions

Jack Langerman

Submit answers only to problems 1, 3 and 4(b) and (c). You do not need to answer 2 or 4(a). But, make sure you know how to do all the problems.

1. Suppose we want to fit a model,

$$\hat{y} = \frac{1}{w_0 + \sum_{j=1}^{d} w_j x_j},$$

for parameters $\mathbf{w}$. Given training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, a nonlinear least squares fit could use the loss function,

$$J(\mathbf{w}) = \sum_{i=1}^{n} \left[y_i - \frac{1}{w_0 + \sum_{j=1}^{d} w_j x_{ij}} \right]^2$$

(a) Find a function $g(\mathbf{z})$ and matrix $\mathbf{A}$ such that the loss function is given by,

$$J(\mathbf{w}) = g(\mathbf{z}), \quad \mathbf{z} = \mathbf{A}\mathbf{w},$$

and $g(\mathbf{z})$ is factorizable, meaning $g(\mathbf{z}) = \sum_i g_i(z_i)$ for some functions $g_i(z_i)$.

$$\mathbf{A} = \begin{bmatrix} — & \mathbf{x}^{(1)T} & — \\ — & \mathbf{x}^{(2)T} & — \\ & \vdots & \\ — & \mathbf{x}^{(n)T} & — \end{bmatrix} \qquad \mathbf{x}^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_d^{(i)} \end{bmatrix}$$

$$g_i(z_i) = \left(y_i - \frac{1}{z_i} \right)^2 \qquad \mathbf{z} = \mathbf{A}\mathbf{w}, \quad z_i = \mathbf{x}^{(i)T}\mathbf{w}$$

(b) What is the gradient $\nabla J(\mathbf{w})$?

$$\frac{\partial g_i(z_i)}{\partial z_i} = 2 \left(\frac{1}{z_i} - y_i \right) \cdot \frac{1}{z_i^2}$$

$$\frac{\partial z_i}{\partial w_j} = \mathbf{x}_j^{(i)}$$

$$\begin{aligned}\nabla J(\mathbf{w})_j &= \sum_{i=1}^{n} \frac{\partial g_i(z_i)}{\partial z_i} \cdot \frac{\partial z_i}{\partial w_j}, \quad j = 0..d \\ &= \sum_{i=1}^{n} \cdot 2\left(\frac{1}{z_i} - y_i\right) \cdot \frac{1}{z_i^2} \cdot \mathbf{x}_j^{(i)}, \quad j = 0..d\end{aligned}$$

(c) What is the gradient descent update for $\mathbf{w}$?

$$\mathbf{w}_i := \mathbf{w}_i - \alpha \sum_{j=1}^{n} \frac{\partial}{\partial \mathbf{w}_i} g_j(z_j), \quad i = 0...d$$

$$\mathbf{w}_i := \mathbf{w}_i - \alpha \left(2 \sum_{j=1}^{n} \frac{1/z_j - y_j}{z_j^2} \cdot x_i^{(j)}\right), \quad i = 0...d$$

(d) Write a few lines of python code to compute the loss function $J(\mathbf{w})$ and $\nabla J(\mathbf{w})$.

```python
def costGrad(A, w, y):
    z = np.matmul(A, w)
    a = 1/z
    J = np.sum((y-a)**2)

    dJda = 2*(a-y)
    dadz = -1/(z**2)
    dzdw = A.T

    grad = np.matmul(dzdw, dJda*dadz) # dJdw

    return J, grad
```

2. In this problem, we will see why gradient descent can often exhibit very slow convergence, even on apparently simple functions. Consider the objective function,

$$J(\mathbf{w}) = \frac{1}{2}b_1 w_1^2 + \frac{1}{2}b_2 w_2^2,$$

defined on a vector $\mathbf{w} = (w_1, w_2)$ with constants $b_2 > b_1 > 0$.

(a) What is the gradient $\nabla J(\mathbf{w})$?

(b) What is the minimum $\mathbf{w}^* = \arg\min_{\mathbf{w}} J(\mathbf{w})$?

(c) Part (b) shows that we can minimize $J(\mathbf{w})$ easily by hand. But, suppose we tried to minimize it via gradient descent. Show that the gradient descent update of $\mathbf{w}$ with a step-size α has the form,

$$w_1^{k+1} = \rho_1 w_1^k, \quad w_2^{k+1} = \rho_2 w_2^k,$$

for some constants ρ_i, $i = 1, 2$. Write ρ_i in terms of b_i and the step-size α.

(d) For what values α will gradient descent converge to the minimum? That is, what step sizes guarantee that $\mathbf{w}^k \to \mathbf{w}^*$.

(e) Take $\alpha = 2/(b_1 + b_2)$. It can be shown that this choice of α results in the fastest convergence. You do not need to show this. But, show that with this selection of α,

$$\|\mathbf{w}^k\| = C^k \|\mathbf{w}^0\|, \quad C = \frac{\kappa - 1}{\kappa + 1}, \quad \kappa = \frac{b_2}{b_1}.$$

The term κ is called the *condition number*. The above calculation shows that when κ is very large, $C \approx 1$ and the convergence of gradient descent is slow. In general, gradient descent performs poorly when the problems are ill-conditioned like this.

3. *Matrix minimization.* Consider the problem of finding a matrix $\mathbf{P} \in \mathbb{R}^{m \times m}$ to minimize the loss function,

$$J(\mathbf{P}) = \sum_{i=1}^{n} \left[\frac{z_i}{y_i} - \ln(z_i) \right], \quad z_i = \mathbf{x}_i^{\mathsf{T}} \mathbf{P} \mathbf{x}_i.$$

The problem arises in wireless communications where an m-antenna receiver wishes to estimate a spatial covariance matrix $\mathbf{P}$ from n power measurements. In this setting, $y_i > 0$ is the i-th receive power measurement and $\mathbf{x}_i$ is the beamforming direction for that measurement. In reality, the quantities would be complex, but for simplicity we will just look at the real-valued case. See the following article for more details:

> Eliasi, Parisa A., Sundeep Rangan, and Theodore S. Rappaport. "Low-rank spatial channel estimation for millimeter wave cellular systems," *IEEE Transactions on Wireless Communications* 16.5 (2017): 2748-2759.

(a) What is the gradient $\nabla_{\mathbf{P}} z_i$?

$$\nabla_{\mathbf{P}} z_i = \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_i$$

(b) What is the gradient $\nabla_{\mathbf{P}} J(\mathbf{P})$?

$$\nabla_{\mathbf{P}} J(\mathbf{P}) = \begin{bmatrix} \frac{\partial J(\mathbf{P})}{\partial \mathbf{P}_{11}} & \frac{\partial J(\mathbf{P})}{\partial \mathbf{P}_{12}} & \cdots & \frac{\partial J(\mathbf{P})}{\partial \mathbf{P}_{1m}} \\ \frac{\partial J(\mathbf{P})}{\partial \mathbf{P}_{21}} & \frac{\partial J(\mathbf{P})}{\partial \mathbf{P}_{22}} & \cdots & \frac{\partial J(\mathbf{P})}{\partial \mathbf{P}_{2m}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial J(\mathbf{P})}{\partial \mathbf{P}_{m1}} & \frac{\partial J(\mathbf{P})}{\partial \mathbf{P}_{m2}} & \cdots & \frac{\partial J(\mathbf{P})}{\partial \mathbf{P}_{mm}} \end{bmatrix}$$

$$\nabla_{\mathbf{P}} J(\mathbf{P})_{ij} = \frac{\partial J(\mathbf{P})}{\partial \mathbf{P}_{ij}} = \frac{\partial J(\mathbf{P})}{\partial z_i} \cdot \frac{\partial z_i}{\partial \mathbf{P}_{ij}}$$

$$= \sum_{i=1}^{n} \left[\frac{1}{y_i} - \frac{1}{z_i} \right] \cdot \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_i$$

(c) Write a few lines of python code to evaluate $J(\mathbf{P})$ and $\nabla_{\mathbf{P}} J(\mathbf{P})$ given data $\mathbf{x}_i$ and y_i. You can use a for loop.

```python
def loss(A, P, y):
        n = A.shape[0]
        z = np.zeros(n)
        dadz = np.zeros(n)
        for i in range(n):
                x = A[i, :].T
                z[i] = np.matmul( np.matmul(x.T, P), x)
                a[i] = (z[i] / y[i]) - np.log(z[i])

                dadz[i] = (1/y[i] - 1/z[i])
                dzdx[i] = np.matmul(x.T, x)
                dJdx[i] = dadz[i]*dzdx[i]

        J = np.sum(a)
        grad = np.sum(dJdx)

        return J, grad
```

(d) See if you can rewrite (c) without a for loop. You will need Python broadcasting.

```python
def loss(A, P, y):
        z = np.matmul( np.matmul(A, P), A.T)
        a = (z / y) - np.log(z)

        dadz = 1/y - 1/z
        dzdx = np.matmul(A, A.T)
        dJdx = dadz * dadz

        J = np.sum(a)
        grad = np.sum(dJdx)

        return J, grad
```

4. *Nested optimization.* Suppose we are given a loss function $J(\mathbf{w}_1, \mathbf{w}_2)$ with two parameter vectors $\mathbf{w}_1$ and $\mathbf{w}_2$. In some cases, it is easy to minimize over one of the sets of parameters, say $\mathbf{w}_2$, while holding the other parameter vector (say, $\mathbf{w}_1$) constant. In this case, one could perform the following *nested* minimization: Define

$$J_1(\mathbf{w}_1) := \min_{\mathbf{w}_2} J(\mathbf{w}_1, \mathbf{w}_2), \quad \widehat{\mathbf{w}}_2(\mathbf{w}_1) := \arg\min_{\mathbf{w}_2} J(\mathbf{w}_1, \mathbf{w}_2),$$

which represent the minimum and argument of the loss function over $\mathbf{w}_2$ holding $\mathbf{w}_1$ constant. Then,

$$\widehat{\mathbf{w}}_1 = \arg\min_{\mathbf{w}_1} J_1(\mathbf{w}_1) = \arg\min_{\mathbf{w}_1} \min_{\mathbf{w}_2} J(\mathbf{w}_1, \mathbf{w}_2).$$

Hence, we can find the optimal $\mathbf{w}_1$ by minimizing $J_1(\mathbf{w}_1)$ instead of minimizing $J(\mathbf{w}_1, \mathbf{w}_2)$ over $\mathbf{w}_1$ and $\mathbf{w}_2$.

(a) Show that the gradient of $J_1(\mathbf{w}_1)$ is given by

$$\nabla_{\mathbf{w}_1} J_1(\mathbf{w}_1) = \left. \nabla_{\mathbf{w}_1} J(\mathbf{w}_1, \mathbf{w}_2) \right|_{\mathbf{w}_2 = \widehat{\mathbf{w}}_2}.$$

Thus, given $\mathbf{w}_1$, we can evaluate the gradient from (i) solve the minimization $\widehat{\mathbf{w}}_2 := \arg\min_{\mathbf{w}_2} J(\mathbf{w}_1, \mathbf{w}_2)$; and (ii) take the gradient $\nabla_{\mathbf{w}_1} J(\mathbf{w}_1, \mathbf{w}_2)$ and evaluate at $\mathbf{w}_2 = = \widehat{\mathbf{w}}_2$.

(b) Suppose we want to minimize a nonlinear least squares,

$$J(\mathbf{a}, \mathbf{b}) := \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{d} b_j e^{-a_j x_i} \right)^2,$$

over two parameters $\mathbf{a}$ and $\mathbf{b}$. Given parameters $\mathbf{a}$, describe how we can minimize over $\mathbf{b}$. That is, how can we compute,

$$\hat{\mathbf{b}} := \arg\min_{\mathbf{b}} J(\mathbf{a}, \mathbf{b}).$$

first setup a matrix $\mathbf{A}$ and parameter vector β such that

$$\mathbf{A} = \begin{bmatrix} e^{-a_1 x_1} & e^{-a_2 x_1} & \dots & e^{-a_d x_1} \\ e^{-a_1 x_2} & e^{-a_2 x_2} & \dots & e^{-a_d x_2} \\ \vdots & \vdots & \ddots & \vdots \\ e^{-a_1 x_n} & e^{-a_2 x_n} & \dots & e^{-a_d x_n} \end{bmatrix} \qquad \beta = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_d \end{bmatrix}$$

$$\mathbf{A}\beta = \mathbf{y}$$

Then solve for β using least squares.

(c) In the above example, how would we compute the gradients,

$$\nabla_{\mathbf{a}} J(\mathbf{a}, \mathbf{b}).$$

$$\nabla_{\mathbf{a}} J(\mathbf{a}, \mathbf{b}) = 2 \sum_{i=1}^{n} \left[\left(y_i - \sum_{j=1}^{d} b_j e^{-a_j x_i} \right) \sum_{j=1}^{d} b_j x_i e^{-a_j x_i} \right]$$