Introduction to Machine Learning
Unit 7 Problem Solutions: Gradient Calculations and Nonlinear Optimization

Prof. Sundeep Rangan

1. *Simple gradient calculation.* Consider a function,

$$J = z_1 e^{z_1 z_2}, \quad z_1 = a_1 w_1 w_2, \quad z_2 = a_2 w_1 + a_3 w_2^2,$$

   (a) Compute the partial derivatives, $\partial J / \partial w_j$ for $j = 1, 2$.

   (b) Write a python function that, given $\mathbf{w}$ and $\mathbf{a}$ computes $J(\mathbf{w})$ and $\nabla J(\mathbf{w})$.

Solution

(a) We first compute the partial derivatives,

$$\frac{\partial J}{\partial z_1} = e^{z_1 z_2} + z_1 z_2 e^{z_1 z_2}, \quad \frac{\partial J}{\partial z_2} = z_1^2 e^{z_1 z_2}.$$

and

$$\begin{aligned} \frac{\partial z_1}{\partial w_1} &= a_1 w_2, \quad \frac{\partial z_1}{\partial w_2} = a_1 w_1, \\ \frac{\partial z_2}{\partial w_1} &= a_2, \quad \frac{\partial z_2}{\partial w_2} = 2 a_3 w_2 \end{aligned}$$

Then, we can use chain rule,

$$\begin{aligned} \frac{\partial J}{\partial w_1} &= \frac{\partial J}{\partial z_1}\frac{\partial z_1}{\partial w_1} + \frac{\partial J}{\partial z_2}\frac{\partial z_2}{\partial w_1}, \\ \frac{\partial J}{\partial w_2} &= \frac{\partial J}{\partial z_1}\frac{\partial z_1}{\partial w_2} + \frac{\partial J}{\partial z_2}\frac{\partial z_2}{\partial w_2}. \end{aligned}$$

You do not to simplify this any further.

(b) Taking the equations above, we can write the function as follows. Note that indexing starts at `0`.

```
def Jeval(w, a):

    # Compute z as a function of w
    z = np.zeros(2)
    z[0] =  a[0]*w[0]*w[1],
    z[1] =  a[1]*w[0] + a[2]*(w[1]**2)
```

```
    # Compute J in terms of z
    J = z[0]*np.exp(z[0]*z[1])

    # Compute dJ/dz
    dJ_dz0 = (z[0]*z[1]+1)np.exp(z[0]*z[1])
    dJ_dz1 = (z[0]**2)*np.exp(z[0]*z[1])

    # Compute dz/dw
    dz0_dw0 = a[0]*w[1]
    dz0_dw1 = a[0]*w[0]
    dz1_dw0 = a[1]
    dz1_dw1 = 2*a[2]*w[1]

    # Compte Jgrad with chain rule
    dJ_dw0 = dJ_dz0*dz0_dw0 + dJ_dz1*dz1_dw0
    dJ_dw1 = dJ_dz0*dz0_dw1 + dJ_dz1*dz1_dw1
    Jgrad = np.array([dJ_dw0, dJ_dw1])

    return J, Jgrad
```

2. *Gradient with a logarithmic loss.* Consider the loss function,

$$J(\mathbf{w}, b) := \sum_{i=1}^{N} (\log(y_i) - \log(\hat{y}_i))^2, \quad \hat{y}_i = \sum_{j=1}^{p} x_{ij} w_j + b,$$

This is an MSE loss function, but in log domain.

(a) Find the gradient components, $\partial J / \partial w_j$ and $\partial J / \partial b$.

(b) Complete the following python function

```
def Jeval(w,b,...):
    ...
    return J, Jgradw, Jgradb
```

that computes J and $\nabla_w J$ and $\nabla_b J$. You need to complete the arguments of the function. To receive full credit, avoid using for loops.

Solution

(a) This is a direct application of chain rule,

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \sum_{i=1}^{N} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial w_j} = 2 \sum_{i=1}^{N} (\log(\hat{y}_i - \log(y_i)) \frac{1}{y_i} x_{ij} \\
\frac{\partial J}{\partial b} &= \sum_{i=1}^{N} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial b} = 2 \sum_{i=1}^{N} (\log(\hat{y}_i - \log(y_i)) \frac{1}{\hat{y}_i}.
\end{aligned}$$

This answer will receive full credit. But, for implementation in the next part, it is convenient to define,

$$q_i = 2(\log(\hat{y}_i - \log(y_i))\frac{1}{y_i}.$$

Then,

$$\frac{\partial J}{\partial w_j} = \sum_{i=1}^{N} x_{ij} q_i, \quad \frac{\partial J}{\partial b} = \sum_{i=1}^{N} q_i.$$

(b) This can be implemented as follows:

```
def Jeval(w,b,X,y):
    yhat = X.dot(w) + b
    yerr = np.log(yhat)-np.log(y)
    q = 2*yerr/yhat
    Jgrad_w = X.T.dot(q)
    Jgrad_b = np.sum(q)
    return J, Jgrad_w, Jgrad_b
```

3. *Gradient with an inverse function.* Consider the nonlinear least squares fit loss funciton

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

(a) Compute the gradient components, $\partial J/\partial w_j$. You may want to define the intermediate variable,

$$z_i = w_0 + \sum_{j=1}^{d} w_j x_{ij}.$$

Also, you can write separate answers for $\partial J/\partial w_0$ and $\partial J/w_j$ for $j = 1, \ldots, d$.

(b) Complete the following function to compute the loss and gradient,

```
def Jeval(w,...):
    ...
    return J, Jgrad
```

For the gradient, you may wish to use the function,

```
Jgrad = np.hstack((Jgrad0, Jgrad1))
```

to stack two vectors or a vector and a scalar.

Solution

(a) If we define, z_i as suggested, the loss function is,

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

Now, we apply chain rule. For $j = 1, \ldots, d$,

$$\frac{\partial J}{\partial w_j} = \sum_{i=1}^{N} \frac{\partial J}{\partial z_i} \frac{\partial z_i}{\partial w_j} = -2 \sum_{i=1}^{N} \left[\frac{1}{z_i} - y_i \right] \frac{1}{z_i^2} \frac{\partial z_i}{\partial w_j} = -2 \sum_{i=1}^{N} \left[\frac{1}{z_i} - y_i \right] \frac{1}{z_i^2} x_{ij}.$$

For $j = 0$,

$$\frac{\partial J}{\partial w_j} = \sum_{i=1}^{N} \frac{\partial J}{\partial z_i} \frac{\partial z_i}{\partial w_j} = -2 \sum_{i=1}^{N} \left[\frac{1}{z_i} - y_i \right] \frac{1}{z_i^2} \frac{\partial z_i}{\partial w_j} = -2 \sum_{i=1}^{N} \left[\frac{1}{z_i} - y_i \right] \frac{1}{z_i^2}.$$

As in the previous problem, you can define,

$$q_i = -2 \left[\frac{1}{z_i} - y_i \right] \frac{1}{z_i^2}.$$

Then,

$$\frac{\partial J}{\partial w_j} = \sum_{i=1}^{N} x_{ij} q_i, \quad \frac{\partial J}{\partial b} = \sum_{i=1}^{N} q_i.$$

(b) This can be implemented as follows. Note that in the beginning we "unpack" the vector `w` into `w[0]` and `w[1:]`. Then, we compute the gradient of each of these terms and stack then with the command `hstack`.

```
def Jeval(w,X,y):
    # Unpack the variables
    w0 = w[0]
    w1 = w[1:]

    # Compute the loss
    z = X.dot(w1) + w0
    J = np.sum((y - 1/z)**2)

    # Compute the gradients
    q = -2*(1/z-y)/(z**2)
    Jgrad0 = np.sum(q)
    Jgrad1 = X.T.dot(q)

    # Concatenate the two gradients
    Jgrad = np.hstack((Jgrad0, Jgrad1))
    return J, Jgrad
```

4. *Gradient with nonlinear parametrization.* Given data (x_i, y_i) with binary class labels $y_i \in \{0, 1\}$, consider the binary cross-entropy loss function,

$$J(\mathbf{a}, \mathbf{b}) := \sum_{i=1}^{N} \log(1 + e^{z_i}) - y_i z_i, \quad z_i = \sum_{j=1}^{d} a_j e^{-(x_i - b_j)^2/2}.$$

(a) Compute the gradient components, $\partial J/\partial a_j$ and $\partial J/\partial b_j$.

(b) Complete the following function to compute the loss and gradient,

```
def Jeval(w,...):
    ...
    return J, Jgrada, Jgradb
```

Avoid for loops to receive full credit.

Solution

(a) Let

$$J_i = \log(1 + e^{z_i}) - y_i z_i,$$

so

$$J = \sum_{i=1}^{N} J_i.$$

We apply chain rule.

$$\frac{\partial J}{\partial a_j} = \sum_{i=1}^{N} \frac{\partial J}{\partial J_i}\frac{\partial J_i}{\partial z_i}\frac{\partial z_i}{\partial a_j} = \sum_{i=1}^{N}\left[\frac{e^{z_i}}{1+e^{z_i}} - y_i\right] e^{-(x_i-b_j)^2/2}$$
$$\frac{\partial J}{\partial b_j} = \sum_{i=1}^{N} \frac{\partial J}{\partial J_i}\frac{\partial J_i}{\partial z_i}\frac{\partial z_i}{\partial b_j} = \sum_{i=1}^{N}\left[\frac{e^{z_i}}{1+e^{z_i}} - y_i\right] a_j(b_j - x_i) e^{-(x_i-b_j)^2/2}.$$

The above answer will receive full credit. But, for implementation, is convenient to define

$$p_i = \frac{e^{z_i}}{1+e^{z_i}} - y_i, \quad Q_{ij} := e^{-(x_i-b_j)^2/2}.$$

Then,

$$\frac{\partial J}{\partial a_j} == \sum_{i=1}^{N} p_i Q_{ij}, \quad \frac{\partial J}{\partial b_j} = \sum_{i=1}^{N} p_i a_j (b_j - x_i) Q_{ij}.$$

(b) The following code implements the above calculations using python broadcasting.

```
def Jeval(a,b,x,y):

    # D[i,j] = x[i] - b[j]
    # Q[i,j] = exp(-D[i,j]**2/2)
    D = x[:,None] - b[None,:]
    Q = np.exp(-D**2/2)

    # z[i] = \sum_j Q[i,j]a[j]
    # J = \sum_i log(1+exp(z[i])) - z[i]*y[i]
    z = Q.dot(a)
    J = np.sum(np.log(1+np.exp(z)) - z*y)

    # p[i] = 1/(1+exp(-z[i])) - y[i]
```

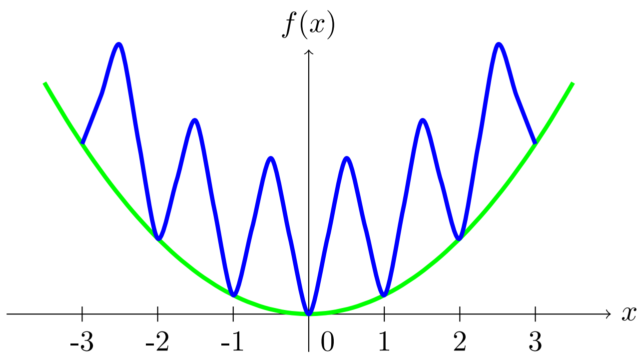

Figure 1: Objective function

```
p = 1/(1+np.exp(-z)) - y

# Gradients
Jgrada = Q.T.dot(p)
Jgradb = -np.sum(a[None,:]*Q*D,axis=0)

return J, Jgrada, Jgradb
```

5. *Finding local and global minima.* Consider the function

$$f(x) = \frac{1}{4}x^2 + 1 - \cos(2\pi x).$$

(a) Approximately draw $f(x)$.

(b) Write an equation for the gradient descent update to minimize $f(x)$.

(c) Using the graph in part (a), where is the global minima of $f(x)$?

(d) Using the graph in part (a), find one initial condition where gradient descent could end up converging to a local minima that is not the global minima. The local minima do not have a closed form expression, but you should be able to use the graph in part (a) to "eyeball" an initial condition close to a bad local minima.

Solution

(a) To plot the function, it is useful to first plot $\frac{1}{4}x^2$, which is the green line in Fig. 1. The function $f(x)$ is the sum of $\frac{1}{4}x^2$ with $1 + \cos(2\pi x)$ which creates the oscillations on top of $\frac{1}{4}x^2$.

(b) We take the derivative,

$$f'(x) = \frac{x}{2} + 2\pi \sin(x),$$

so the gradient descent with step size $\alpha > 0$ is

$$x_{k+1} = x_k - \alpha \left[\frac{x_k}{2} + 2\pi \sin(x_k) \right].$$

(c) At all critical points,

$$f'(x) = \frac{x}{2} + 2\pi \sin(x) = 0 \Rightarrow -\frac{x}{4\pi} = \sin(x). \tag{1}$$

One solution to this equation is at $x = 0$. From Fig. 1, you can see that this is the global minima.

(d) The other solutions to (1) do not have a closed form expression, but we can see from Fig. 1, that there is a local minima close to $x = 1$. Thus, if we start gradient descent close to $x_0 = 1$ with a small step size, it will converge to the local minima.

6. *Rate of convergence and condition number.* 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.

Solution

(a) The gradient $\nabla J(\mathbf{w})$,

$$\nabla J(\mathbf{w}) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}\right]^\mathsf{T} = [b_1 w_1, b_2 w_2]^\mathsf{T}.$$

(b) Set $\nabla J(\mathbf{w}) = 0$, then we get $\mathbf{w}^* = 0$.

(c) We have

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \alpha \nabla J(\mathbf{w}^k) \Rightarrow w_i^{k+1} = w_i^k - \alpha b_i w_i^k = \rho_i w_i^k, \tag{2}$$

where $\rho_i = 1 - b_i\alpha$.

(d) In order that $\mathbf{w}^k \to \mathbf{w}^* = 0$, we need $|\rho_i| < 1$ for $i = 1, 2$. Since $b_i > 0$ and $\alpha > 0$, we need

$$|1 - b_i\alpha| < 1 \Rightarrow \alpha < \frac{2}{b_i},$$

for $i = 1, 2$.

(e) For $\alpha = 2/(b_1 + b_2)$, we have

$$\rho_1 = 1 - b_1\alpha = \frac{b_2 - b_1}{b_1 + b_2}, \quad \rho_2 = 1 - b_2\alpha = \frac{b_1 - b_2}{b_1 + b_2}.$$

Hence, if we set

$$C = \frac{b_2 - b_1}{b_1 + b_2},$$

we have $|\rho_i| = C$ for $i = 1, 2$. Also, since $\kappa = b_2/b_1$,

$$C = \frac{\kappa - 1}{\kappa + 1}.$$

Now, from (2), $w_i^k = \rho_i^k w_i^0$ so $|w_i^k| = C|w_i^0|$. Therefore,

$$\|\mathbf{w}^k\|^2 = |w_1^k|^2 + |w_2^k|^2 = C^{2k}\left[|w_1^0|^2 + |w_2^0|^2\right] = C^{2k}\|\mathbf{w}^0\|^2.$$

This shows $\|\mathbf{w}^k\| = C\|\mathbf{w}^0\|$.

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

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

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 $\nabla_{\mathbf{P}} z_i$?

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

(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.

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

Solution

(a) First observe that

$$z_i = \mathbf{x}_i^\mathsf{T} \mathbf{P} \mathbf{x}_i = \sum_{j,k} x_{ij} x_{ik} P_{jk} \Rightarrow \frac{\partial z_i}{\partial P_{jk}} = x_{ij} x_{ik}.$$

So $\nabla_{\mathbf{P}} z_i$ is the matrix with elements $x_{ij} x_{ik}$. Therefore,

$$\nabla_{\mathbf{P}} z_i = [x_{ij} x_{ik}]_{jk} = \mathbf{x}_i \mathbf{x}_i^\mathsf{T}.$$

(b) By the chain rule,

$$\frac{\partial J}{\partial P_{jk}} = \sum_{i=1}^{n} \frac{\partial J}{\partial z_i} \frac{\partial z_i}{\partial P_{jk}} = \sum_{i=1}^{n} \left[\frac{1}{y_i} - \frac{1}{z_i} \right] \frac{\partial z_i}{\partial P_{jk}}.$$

Therefore,

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

(c) We can first write this with a for loop as follows:

```
# Compute z
n = X.shape[0]
z = np.zeros(n)
for i in range(n):
    z[i] = X[i,:].dot(P.dot(X[i,:])

# Compute J
J = np.sum(z/y - np.log(z))
g = 1/y - 1/z

# Compute gradient
Jgrad = np.zeros((n,n))
for i in range(n):
    xi = X[i,:]
    Jgrad += g[i]*xi[:,None]*xi[None,:]
```

Note the use of `xi[:,None]*xi[None,:]` to compute $\mathbf{x}_i \mathbf{x}_i^\mathsf{T}$.

(d) To remove the for-loops, we can use the following code:

```
# Compute z
XP = X.dot(P)
z = np.sum(XP*X, axis=1)

# Compute J
J = np.sum(z/y - np.log(z))
g = 1/y - 1/z
```

```
# Compute gradient
GX = g[:,None]*X
Jgrad = X.T.dot(GX)
```

To understand this code, first observe that the i-th row of the matrix `XP` is simply $\mathbf{x}_i^\mathsf{T}\mathbf{P}$. Thus, element (i,j) of `XP*X` is

$$(\mathbf{x}_i^\mathsf{T}\mathbf{P})_j x_{ij}.$$

When we sum this over `axis=1`, we obtain the sum,

$$\sum_{j=1}^{d}(\mathbf{x}_i^\mathsf{T}\mathbf{P})_j x_{ij} = \mathbf{x}_i^\mathsf{T}\mathbf{P}\mathbf{x}_i = z_i.$$

Hence, we obtain `z = np.sum(XP*X, axis=1)`. For the gradient, note that element (i,j) of `GX` is $g_i x_{ij}$, where

$$g_i = \frac{1}{y_i} - \frac{1}{z_i}.$$

Therefore, the (k,j) element of `Jgrad` is

$$\sum_{i=1}^{n} x_{ik} g_i x_{ij} = \sum_{i=1}^{n} g_i \left[\mathbf{x}_i\mathbf{x}_i^\mathsf{T}\right]_{kj},$$

and hence `Jgrad` is the matrix,

$$\sum_{i=1}^{n} g_i \mathbf{x}_i\mathbf{x}_i^\mathsf{T} = \nabla_{\mathbf{P}} J(\mathbf{P}).$$

8. *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

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

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,

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

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

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

Solution

(a) Since $\widehat{\mathbf{w}}_2 = \arg\min_{\mathbf{w}_2} J(\mathbf{w}_1, \mathbf{w}_2)$, we have

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

To take the derivative with respect to $\mathbf{w}_1$ we must remember that $\widehat{\mathbf{w}}_2$ is a function of $\mathbf{w}_1$. Therefore,

$$\begin{aligned} \frac{\partial J_1}{\partial w_{1j}} &= \frac{\partial J(\mathbf{w}_1, \widehat{\mathbf{w}}_2)}{\partial w_{1j}} \\ &= \left. \frac{\partial J(\mathbf{w}_1, \mathbf{w}_2)}{\partial w_{1j}} \right|_{\mathbf{w}_2 = \widehat{\mathbf{w}}_2} + \sum_k \left. \frac{\partial J(\mathbf{w}_1, \mathbf{w}_2)}{\partial w_{2k}} \right|_{\mathbf{w}_2 = \widehat{\mathbf{w}}_2} \frac{\partial w_{2k}}{\partial w_{1j}} \end{aligned} \tag{3}$$

Now, since $\widehat{\mathbf{w}}_2$ minimizes $J(\mathbf{w}_1, \mathbf{w}_2)$ over $\mathbf{w}_2$, we must have

$$\nabla_{\mathbf{w}_2} J(\mathbf{w}_1, \mathbf{w}_2)\big|_{\mathbf{w}_2 = \widehat{\mathbf{w}}_2} = 0 \Rightarrow \left. \frac{\partial J(\mathbf{w}_1, \mathbf{w}_2)}{\partial w_{2k}} \right|_{\mathbf{w}_2 = \widehat{\mathbf{w}}_2} = 0,$$

for all k. Therefore, from (3),

$$\frac{\partial J_1}{\partial w_{1j}} = \frac{\partial J(\mathbf{w}_1, \widehat{\mathbf{w}}_2)}{\partial w_{1j}} \Rightarrow \nabla_{\mathbf{w}_1} J_1(\mathbf{w}_1) = \nabla_{\mathbf{w}_1} J(\mathbf{w}_1, \mathbf{w}_2)\big|_{\mathbf{w}_2 = \widehat{\mathbf{w}}_2}.$$

(b) When $\mathbf{a}$ is fixed, the minimization over $\mathbf{b}$ is a linear least squares problem. To see this, let

$$\hat{y}_i = \sum_{j=1}^{d} b_j e^{-a_j x_i}, \tag{4}$$

so we can write the loss function as

$$J(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

Thus, $J(\mathbf{a}, \mathbf{b})$ is exactly the RSS. Also, we have $\hat{\mathbf{y}} = \mathbf{A}\mathbf{b}$ where $\mathbf{A}$ is the matrix

$$\mathbf{A} = \begin{bmatrix} e^{-a_1 x_1} & \cdots & e^{-a_d x_1} \\ \vdots & \cdots & \vdots \\ e^{-a_1 x_n} & \cdots & e^{-a_d x_n} \end{bmatrix}.$$

It follows that the optimal $\mathbf{b}$ is given by the least-squares formula

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b}} J(\mathbf{a}, \mathbf{b}) = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{y}.$$

(c) The partial derivative

$$\begin{aligned} \frac{\partial J(\mathbf{a}, \mathbf{b})}{\partial a_j} &= \sum_{i=1}^{n} \frac{\partial (y_i - \hat{y}_i)^2}{\partial a_j} = -2 \sum_{i=1}^{n} (y_i - \hat{y}_i) \frac{\partial \hat{y}_i}{\partial a_j} \\ &= 2 \sum_{i=1}^{n} (y_i - \hat{y}_i) b_j x_i e^{-a_j x_i}. \end{aligned} \tag{5}$$

The gradient is

$$\nabla_{\mathbf{a}} J(\mathbf{a}, \mathbf{b}) = \left[\frac{\partial J(\mathbf{a}, \mathbf{b})}{\partial a_1}, \cdots, \frac{\partial J(\mathbf{a}, \mathbf{b})}{\partial a_d} \right]^\top.$$