Introduction to Machine Learning

Homework 5: Gradient Calculations and Nonlinear Optimization

[AUTHOR]

Student Name: Can Xu

Net id: cx461

Community Mail: jonathanxqs@nyu.edu

[Solution]

1.

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 $g_i(z_i)$.

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

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

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

(a) g(**z**)= $\sum_{k=1}^{n} [y_k - {z_k}^{-1}]^2 = \sum_{k=1}^{n} g(z_k)$

A=[[A11, A12, ….,A1d]

… … … …

[An1, An2, ….,And]]

(b)

$$(2) = \nabla_w J(w) = \begin{bmatrix} \partial J(\vec{w}) / \partial w_0 \\ \partial J(\vec{w}) / \partial w_1 \\ \vdots \\ \dfrac{\partial J(\vec{w})}{\partial w_d} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} 2\left(y_i - \frac{1}{w_0+\sum w_j x_{ij}}\right) \cdot \frac{1}{(w_0+\sum w_j x_{ij})^2} \\ \sum_{i=1}^{n} 2\left(y_i - \frac{1}{w_0+\sum}\right) \cdot \frac{x_{i1}}{(w_0+\sum w_j x_{ij})^2} \\ \vdots \\ \sum_{i=1}^{n} 2\left(y_i - \frac{1}{w_0+\sum}\right) \cdot \frac{x_{id}}{(w_0+\sum\ )^2} \end{bmatrix}$$

That is the gradient

(C)

- Gradient descent algorithm:
  - Start with initial w^0
  - $w^{k+1} = w^k - \alpha_k \nabla f(w^k)$
  - Repeat until some stopping criteria
- α_k is called the step size
  - In machine learning, this is called the learning rate

$\nabla f(w^k)$ is known , we can choose α_k as learning rate

(D)

```
y = # yi known from data
tmp = 1 / ( w[0] + x.dot(w[1:]))

J = np.sum ( (y- tmp)**2 )
Jgrad = (y-tmp)* ( (x).dot(tmp**2) )
```

Use boradcasting

3.

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

(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.

(a) ZGrad[i] = x[i].dot(P.dot(x[i].T))

(b) Jgrad = $\sum_{i=1}^{n}$ [ZGrad[i]/y[i] - ZGrad[i]/z[i]]

(n,n)

(c)

```
# x y known
for i in range(n):
    z[i] = x[i].T.dot(P.dot(x[i]))
    zgrad[i] = x[i].dot(P.dot(x[i].T))  # (n,n)
    J =J + (z[i]/y[i] - ln(z[i]) )      # (n,)
    Jgrad += zgrad[i]/y[i] - zgrad[i]/z[i]  # (n,n)
```

(d) broadcasting:

```
J = z[:]/y[:] - ln(z[:])    # (n,)
Jgrad = zgrad[:]/y.reshape(n,-1) - zgrad[:]/z.reshape(n,-1)   # (n,n)
```

Let it be n x n

4.

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 $\mathbf{w}_2$.

(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,

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

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

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

(b) Given **a** ,

let Jgrad(**b**) = $\nabla_{\mathbf{a}} J(\mathbf{a}, \mathbf{b})$. = 0 , solve the related **b**

That is local minima , compare these solutions and get global minima

(c)

a known, b unknown

$$\nabla_a J(a,b) = \begin{bmatrix} \frac{\partial \vec{J}}{\partial b_1} \\ \frac{\partial \vec{J}}{\partial b_2} \\ \vdots \\ \frac{\partial \vec{J}}{\partial b_n} \end{bmatrix} = \begin{bmatrix} 2\sum_{i=1}^{n} (y_i - \sum_{j=1}^{d} b_j e^{-a_j x_i})(-b_1 e^{-a_1 x_i}) \\ \vdots \\ 2\sum_{i=1}^{n} (y_i - \sum_{j=1}^{d} b_j e^{-a_j x_i})(-b_n e^{-a_n x_i}) \end{bmatrix}$$

Let it be = 0