Dorina Ismaili & Deniz Gaiser
04.10.2024

Mining Massive Datasets
Homework 3

Exercise 1

a) function:

$$f = \mathrm{sum}((w_1 + w_2 \cdot x - y)^2)$$

gradient:

$\frac{\partial f}{\partial w_1} = 2 \cdot (w_1 + w_2 \cdot x - y)$
$\frac{\partial f}{\partial w_2} = 2 \cdot (w_1 + w_2 \cdot x - y)^\top \cdot x$

where

- w_1 is a vector
- w_2 is a scalar
- x is a vector
- y is a vector

Explanation on settings: Trivially, x and y were defined as vectors, and w_1, w_2 as scalars. However, the site was not able to add w_1 as a constant to the individual element-wise results, which is, why the w_1 was changed to a vector, with the fact in mind, that all entries in the vector are identical. This then yielded the above results.

b) function:

$$f = \mathrm{sum}((rxi - qi \odot px)^2) + lambda1 \cdot \|px\|_2 + lambda2 \cdot \|qi\|_2$$

gradient:

$\frac{\partial f}{\partial qi} = lambda2/\|qi\|_2 \cdot qi - 2 \cdot (rxi - qi \odot px) \odot px$
$\frac{\partial f}{\partial px} = lambda1/\|px\|_2 \cdot px - 2 \cdot (rxi - qi \odot px) \odot qi$

where

- $lambda1$ is a scalar
- $lambda2$ is a scalar
- px is a vector
- qi is a vector
- rxi is a vector

Exercise 2

a) What is the purpose of automatic differentiation (AD) in computational mathematics, and how does it differ from numerical and symbolic differentiation? Describe at least one issue with numerical differentiation and one with symbolic differentiation that AD resolves.

b) Explain the forward mode and reverse mode of AD. Describe the scenarios where each of these approaches is particularly efficient.

c) How does the time complexity of forward mode compare to reverse mode in terms of gradients for functions with multiple inputs and outputs?

d) In the reverse mode of AD, memory consumption can be a bottleneck. Discuss how checkpointing strategies can mitigate this issue and provide an example of when checkpointing would be essential.

Solution

a) The purpose of automatic differentiation is to compute the derivatives efficiently and we need an automatic approach. AD could be manual, numerical, symbolic and automatic. The numerical differentiation uses the method of finite differences and follows from the limit definition of derivative. For the numeric differentiation an issue could be with accuracy and stability, one is truncation error. We approximate the limit as h goes to 0, where h is non-zero. As h approaches zero, and secant line approaches tangent line, the truncation error decreases leading to rounding error. In ML, some approximation error is ok, but we need O(n) evaluation of a n dimensional gradient. Symbolic differentiation is an automated version of manual differentiation. It allows exact computation and applies standard differentiation rules. One problem is expression swell, where derivative expression may be exponentially longer than original function. It happens when the expression is large, hence leading to duplicated forms on the functions. It requires the function to be expressed in closed form. It operates directly on the program of interest. AD bypasses symbolic differentiation in implementation, and easily handles closed forms.

b) There are two model of auto differentiation: forward mode and reverse mode. Forward mode means augmenting each variable with its derivative. We create a tuple composed of intermediate function called primal paired with its derivative called tangent. We can compute partial derivatives in a single forward pass. We can compute Jacobian, for n¡m. We set x, and proceed with autodiff. There are several ways to implement forward mode. the simplest is to leverage operator overloading, we write a simple class of attributes, and overload it to handle derivative computation. Fey input use forward mode. Meanwhile Reverse mode, allows us to handle cases with many inputs, where the evaluation is computed backwards. This is done in two part process. We start with a forward pass, evaluating intermediate variables, and store the dependencies of expression tree in memory. After completition of the forward pass, we compute the partial derivative of the output with respect to the intermediate variables quantities known as adjoint. To obtain the adjoin we look at each node and take the sum of this produce. In the context of neural network, is a special case of reverse mode autodiff. For general vector valued function, it is computed from row to row.

c) For functions with multiple inputs and outputs, reverse mode produces one row of the Jacobian at a time. A single pass for forward and revere mode does not take much longer than computing the function itself. It takes less that 6 times than number of operations in the original function implementation.

d) Reverse mode is more memory intensive. We have to store the values of intermediate variables and their dependencis in memory. There have been various techniques: Rematerialization where some are stored, and the remainder are recomputed. Usually a hybrid approach is used.

Exercise 3

The following table shows a utility matrix with ratings on a 1-5 star scale of eight items, a through h, by three users A, B, and C.

-	a	b	c	d	e	f	g	h
A	4	5		5	1		3	2
B		3	4	3	1	2	1	
C	2		1	3		4	5	3
D	1		1		4		4	2

a) Compute r_{Ce} using the User-User Collaborative Filtering method. Let $N(e; C)$ be the set of users most similar to C, who have already rated item e. Use $|N(e; C)| = 2$ in your solution, i.e. consider only top-2 most similar users.

b) Repeat Part a) but use the Item-Item Collaborative Filtering method. Let N (e; C) be the set of similar items rated by user C, and use $|N(e; C)| = 2$ in your solution. Compare this result to the one obtained in Part a) and state your conclusions.

Solution

User A: 4, 5 ,0 ,5 ,1 ,0 ,3 ,2
User B:0 ,3, 4, 3,1 ,2 ,1 ,0
User C: 2, 0 ,1 ,3 ,0 ,4 ,5 ,3
User D: 1, 0,1,0,4,0,4,2

a) We calculate the average ratings for each user to normalize ratings: $r_A = 2.5, r_B = 1.75, r_C = 2.25, r_D = 1.5$.
Next, we use cosine similarity to find the similarity between user C and each other user who rated e.
User A and C
1. Dot product: 44
2. $||A|| = 8.9$
3. $||C|| = 8$
4. Cosine Similarity $= \frac{44}{8.9*8} = 0.617$
User B and C
1. Dot product: 26
2. $||B|| = 8.9$
3. $||C|| = 6.32$
4. Cosine Similarity $= \frac{26}{8.9*6.32} = 0.46$
User D and C
1. Dot product: 29
2. $||D|| = 6.16$
3. $||C|| = 8$
4. Cosine Similarity $= \frac{29}{8*6.16} = 0.58$
Based on the similarity we use A and D. To estimate r_C we proceed as follows: $r_{C_e} = \bar{r}_C + \frac{cos(C,D))(r_D - \bar{r}_D) + cos(A,C)(r_A - \bar{R}_A)}{|cos(C,B) + cos(C,A)}|$.
The result is $r_{C_e} = 2.25 + (0.58 * 2.5 + 0.617 * -1.5)/(0.617 + 0.58) = 0.43$ The estimated rating r_C for item e by user C is approximately 0.43.

b) We use Item-item Collaborative Filtering. We check similarity between each elemts. **Similarity e and a**
1. Common users A,D
2. Numerator: 8
3. Denominator: 17
4. Similarity 0.47
Similarity e and b
1. Common users A,B
2. Numerator: 8
3. Denominator: $\sqrt{68}$
4. Similarity 0.97
Similarity e and c
1. Common users B,D
2. Numerator: 8
3. Denominator: 17
4. Similarity 0.47
Similarity e and d
1. Common users A,B
2. Numerator: 8
3. Denominator: $\sqrt{68}$
4. Similarity 0.97
Similarity e and f
1. Common users B
2. Numerator: 2
3. Denominator: 2
4. Similarity 1
Similarity e and g

1. Common users A,B,D
2. Numerator: 20
3. Denominator: 30.2
4. Similarity 0.66

Similarity e and h

1. Common users A,D
2. Numerator: 10
3. Denominator: 11.6
4. Similarity 0.86

Based on the similarity values, the two items most common to e are f and d. Next, we predict the r_C using the ratings and similarities. We know that $r_{C_f} = 4$, $r_{C_d} = 3$. It holds that $r_{C_e} = 3.5$.
In this case, the Item-item Collaborative Filtering methods predicts a higher rating for item e by user C that the User-User method.

Exercise 5

a) Describe each of the following transformations: join(), sortBy(), groupBy(). In particular, what is in each case the type of input and type of output? Give for each one an example ("on paper") with a simple input and output.

b) Give three own programming examples of your choice for transformations (but not for map() or filter()) and three examples for actions (again, of your choice). Write executable code and test its correctness (either single program or several ones). To generate initial RDDs you can use code from lecture one or from Spark documentation. Submit as solution the source code and results of program runs.

Solution

a) join(): returns an RDD containing all pairs of elements with matching keys in self and other. Input type: two RDDs of key-value pairs such as RDD(K,V) and RDD[(K,W)], Output type: An RDD with format RDD[(K,(V,W0))] where each element contains a key and a tuple of values from both input RDDs that matched on that key. Ex: result = rdd1.join(rdd2)
sortBy(): sorts the elements of an RDD based on a specified key. Input type: an RDD of any type RDD[T], with a function that extracts a key for sorting. Output type: Ann RDD sorted based on the key provided. Example: rdd = [('a',3),('b',1),('c',2)], result = rdd.sortBy(lambda x: x[1])
groupBy(): groups elements in ann RDD based on a function that returns a key for grouping. Input type: an RDD of any type RDD[T] with a function that definnnes a key for grouping. Output type: An RDD of key-value pairs, where each key is associated with an iterable collection of elements. Ex: rdd= [1,2,3,4,5,6], result = rdd.groupBy(lambda x: x / 2)

b) The py file.

Exercise 6