Exercise: The Max-Product Algorithm

Bookmark this page

Exercise: The Max-Product Algorithm - Computational Complexity

4/4 points (graded)

Let's figure out the time complexity of the max-product algorithm, i.e., how many operations it takes to run it on a tree with n nodes and where each X_i takes on one of k possible values.

- How many operations does it take to pass max-product messages (including computing/storing traceback tables) from leaves to the arbitrarily chosen root?

  Choose the answer with **smallest** big O bound in terms of k and n (unless one of these doesn't matter).

  - $\mathcal{O}(k)$
  - $\mathcal{O}(k^2)$
  - $\mathcal{O}(k^3)$
  - $\mathcal{O}(nk)$

- (selected) $\mathcal{O}(nk^2)$ ✔
- $\mathcal{O}(nk^3)$

Solution:

The main thing to note is that computing max-product messages from leaves to the root has the same time complexity as computing sum-product messages from leaves to the root, which from the previous exercise on “Speeding Up Sum-Product" we saw takes time $\mathcal{O}(nk^2)$ (note that computing messages from the leaves to the root does not require any modification to the sum-product or the max-product algorithm to produce a running time of $\mathcal{O}(nk^2)$, i.e., the issue we encountered with the star graph only was an issue when we were passing messages from the root back to the leaves).

The reason why the running time is the same as for sum-product specifically for passing messages from the leaves to the root is that we are still iterating over the same number of elements doing maximization instead of summation, and separately we are also storing the argmax for each maximization, which can be done within the same for loop as the maximization itself. To make this clear, we illustrate a code example that shows the difference between computing a sum and computing a maximum where in the latter we keep track of the arg max (note that this is *not* doing sum-product or max-product but something similar to these code examples would go into the sum-product or max-product algorithms):

Summation: Given a dictionary `table` consisting of numbers, we add up to these numbers and store the total in a variable `total`.

```
total = 0
for x in table:
    total += table[x]
```

Maximization keeping track of argmax: Given a dictionary `table` consisting of numbers, we find which of these numbers is largest and what the corresponding key in the dictionary is.

```
maximum = -np.inf
arg_max = None
for x in table:
    if table[x] > maximum:
        maximum = table[x]
        arg_max = x
```

Notice that summation and maximization (keeping track of argmax as well) each takes time that is linear in the number of dictionary keys in `table`.

The main change in going from sum-product to max-product is precisely doing maximization instead of summation and keeping track of the argmax and storing the argmax in a table. The extra storage needed is roughly twice the amount we used before where now we keep track of, for each max-product message, also a traceback table.

- Recall that d_r is the number of neighbors that node r has, also called the degree of node r. How many operations does it take to compute, for the root node, $\widehat{x}_r = \arg\max_{x_r} \phi_r(x_r) \prod_{\ell \in \mathcal{N}(r)} m_{\ell \to r}(x_r)$?

  Choose the answer with **smallest** big O bound in terms of k and d_r (unless one of these doesn't matter).

  - $\mathcal{O}(d_r)$

- $\mathcal{O}(d_r k)$ ✔
- $\mathcal{O}(d_r k^2)$
- $\mathcal{O}(d_r k^3)$

  - This step takes the same amount of time as computing the root node's marginal distribution for the sum-product algorithm. Again, the change is computing the max/argmax which as we explained in the previous part takes the same amount of time as computing a summation in terms of big O.

- How many operations does it take to follow the backpointers?

  Choose the answer with **smallest** big O bound in terms of k and n (unless one of these doesn't matter).

  - $\mathcal{O}(k)$
  - $\mathcal{O}(k^2)$
  - $\mathcal{O}(k^3)$
  - $\mathcal{O}(n)$ ✔

- $\mathcal{O}(nk)$
- $\mathcal{O}(nk^2)$
- $\mathcal{O}(nk^3)$

Solution:

This part is not in sum-product. Following backpointers just involves going from the root to the leaves each time looking up a traceback table entry, so the time complexity is linear in the number of edges, which for a tree is $n-1$. Thus, following backpointers has running time $\mathcal{O}(n)$.

- How many operations does it take to run the max-product algorithm?

- $\mathcal{O}(k)$
- $\mathcal{O}(k^2)$
- $\mathcal{O}(k^3)$
- $\mathcal{O}(nk)$
- $\mathcal{O}(nk^2)$ ✔

- $\mathcal{O}(nk^3)$

Solution:

Putting together the pieces above, max-product has running time $\mathcal{O}(nk^2) + \mathcal{O}(d_r k) + \mathcal{O}(n) = \boxed{\mathcal{O}(nk^2)}$.

Submit You have used 1 of 5 attempts

✔ Correct (4/4 points)

Exercise: The Max-Product Algorithm and the Most Probable Values for Node Marginals

1/1 point (graded)

Warning: There is only 1 attempt for this problem since it is a single yes/no question.

Suppose we run the max-product algorithm to produce

$$(\widehat{x}_1, \widehat{x}_2, \ldots, \widehat{x}_n) = \arg \max_{x_1, x_2, \ldots, x_n} p_{X_1, X_2, \ldots, X_n}(x_1, x_2, \ldots, x_n).$$

Suppose we separately run the sum-product algorithm to produce p_{X_i} for every i, and then we compute

$$\widehat{x}_i^{(\text{sum-product})} = \arg\max_{x_i} p_{X_i}(x_i).$$

Do we always have $\widehat{x}_i = \widehat{x}_i^{(\text{sum-product})}$ for every i?

- Yes
- No ✔

Solution:

$\boxed{\text{No}}$. Counter-example (in red are the marginal distributions, and bolded is the maximum; note that the argmax for node marginals p_{X_1} and p_{X_2} are 0 and 0 respectively, whereas it's clear from looking at the joint probability table that the most probable configuration is actually $X_1 = 1, X_2 = 0$):

p_{X_1,X_2}

X_1 \ X_2	0	1	
0	11/42	11/42	**11/21**
1	12/42	8/42	10/21
	23/42	19/42	

Importantly, we obtain node marginals by summing out other random variables. For max-product, we are doing maximizations instead of summations! A table that is specific to a node that is more along the lines of what max-product is doing is called a max-marginal, which we explore in the next problem.

Submit You have used 1 of 1 attempts

✔ Correct (1/1 point)

Exercise: The Max-Marginals Variant of the Max-Product Algorithm

5/5 points (graded)

In this next problem, we look at a variant of the max-product algorithm that more closely resembles the sum-product algorithm, where we do not keep track of traceback messages. In particular we pass max-product messages from leaves to the arbitrarily chosen root and then pass max-product messages from the root back to the leaves. Finally, for every node i, we compute what is called a node "max-marginal" table $\overline{p}_{X_i}$, which is a table where

$$\overline{p}_{X_i}(x_i) = \phi_i(x_i) \prod_{\ell \in \mathcal{N}(i)} m_{\ell \to i}(x_i),$$

and $m_{\ell \to i}$ here is a max-product message (*not* a sum-product message).

Note that

$$\overline{p}_{X_i}(x_i) \propto \underbrace{\max_{x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n}}_{\text{maximize over everything except } x_i} p_{X_1, X_2, \ldots, X_n}(x_1, x_2, \ldots, x_n).$$

(As a quick check for yourself, be sure you understand why this last proportionality is true! Note that the difference between a node max-marginal $\overline{p}_{X_i}$ and a node marginal p_{X_i} is that a node marginal p_{X_i} corresponds to *summing* the joint distribution over everything except x_i instead of *maximizing* over everything except x_i.)

Now let's examine whether we can get away with not using traceback messages. Consider a two-node graphical model with the pairwise/edge potential table:

$\psi_{1,2}$

X_1 \ X_2	0	1
0	1	10
1	10	1

In this problem, there are no node potentials, so you can think of the node potentials as tables where all the entries are 1.

- Determine the max-marginal tables $\overline{p}_{X_1}$ and $\overline{p}_{X_2}$. For grading purposes, please normalize each of these tables so that they sum to 1. *Note that max-marginal tables in general do not have to sum to 1!*

$\overline{p}_{X_1}(0) =$ 0.5 ✔ **Answer:** 1/2

$\overline{p}_{X_2}(0) =$ 0.5 ✔ **Answer:** 1/2

Solution:

We have

$$\begin{aligned}\overline{p}_{X_1}(x_1) &\propto \underbrace{\phi_1(x_1)}_{1} m_{2\to 1}(x_1) \\ &= m_{2\to 1}(x_1) \\ &= \max_{x_2} \underbrace{\phi_2(x_2)}_{1} \psi_{1,2}(x_1, x_2) \\ &= \max_{x_2} \psi_{1,2}(x_1, x_2) \\ &= \max\{\psi_{1,2}(x_1, 0), \psi_{1,2}(x_1, 1)\}.\end{aligned}$$

In particular, when $x_1 = 0$, we have

$$\overline{p}_{X_1}(0) \propto \max\{\underbrace{\psi_{1,2}(0,0)}_{1}, \underbrace{\psi_{1,2}(0,1)}_{10}\} = 10,$$

and when $x_1 = 1$, we also have

$$\overline{p}_{X_1}(1) \propto \max\{\underbrace{\psi_{1,2}(1,0)}_{10}, \underbrace{\psi_{1,2}(1,1)}_{1}\} = 10.$$

So node max-marginal $\overline{p}_{X_1}$ is a table where both values are the same. Rescaling so that the entries sum to 1 (this is just for inputting the answer; in general, the node max-marginal's entries need not sum to 1), we have $\boxed{\overline{p}_{X_1}(0) = 1/2}$.

One can show with a very similar calculation that the node max-marginal $\overline{p}_{X_2}$ is actually the same as $\overline{p}_{X_1}$, so rescaling so the entries sum to 1, $\boxed{\overline{p}_{X_2}(0) = 1/2}$.

- What value(s) for x_1 has the highest value in the max-marginal table $\overline{p}_{X_1}$, i.e., what is $\arg\max_{x_1} \overline{p}_{X_1}(x_1)$?

  - [x] 0
  - [x] 1

  ✔

  Solution:

  Since the node max-marginal $\overline{p}_{X_1}$ has the same value for both entries in the table, there is no unique argmax: both 0 and 1 are equally good.

- What value(s) for x_2 has the highest value in the max-marginal table $\overline{p}_{X_2}$, i.e., what is $\arg\max_{x_2} \overline{p}_{X_2}(x_2)$?

  - [x] 0
  - [x] 1

  ✔

  Solution:

Since the node max-marginal $\overline{p}_{X_2}$ has the same value for both entries in the table, there is no unique argmax: both 0 and 1 are equally good.

- What value(s) for the pair (x_1, x_2) maximizes the joint probability table p_{X_1,X_2}, i.e., what is $\arg\max_{x_1,x_2} p_{X_1,X_2}(x_1, x_2)$ (you should not have to actually do any computation for this)?

  - [ ] $(0, 0)$
  - [x] $(0, 1)$ ✔
  - [x] $(1, 0)$ ✔
  - [ ] $(1, 1)$

  ✔

  Solution:

  Just by looking at the pairwise/edge potential, it's clear that the two configurations with the highest potential of 10 are $(0, 1)$ and $(1, 0)$.

Importantly: Without keeping track of traceback messages, we would not be able to readily tell which value of x_2 goes with the optimal value of x_1, or vice versa.

At the same time, what this exercise suggests is that we run into a problem when the argmax is not unique. In the practice problem on the next page, we ask you to prove that in fact, if every node max-marginal $\overline{p}_{X_i}$ has a unique argmax $x_i^* = \arg\max_{x_i} \overline{p}_{X_i}(x_i)$, then indeed the unique most probable configuration for the joint probability table $p_{X_1,\ldots,X_n}$ is just $(x_1^*, x_2^*, \ldots, x_n^*)$. This is an important

result because it tells us a specific case in which the direction-free max-product algorithm can be used and produces a correct most probable configuration from just looking at the best configuration per node max-marginal (i.e., where we don't use traceback messages)!

Submit You have used 1 of 5 attempts

✔ Correct (5/5 points)

Discussion

Topic: Inference with Graphical Models - Most Probable Configuration / Exercise: The Max-Product Algorithm

Show Discussion


Download on the
App Store
ANDROID APP ON
Google play