Max-sum algorithm

Method to compute.

\[
x_{\text{max}} = \arg \max_x p(x)
\]

i.e., \( p(x_{\text{max}}) = \max_x p(x) \)

Helpful facts

First, note that.

\[
\max p(x) = \max_{x_1} \max_{x_2} \ldots \max_{x_M} p(x) \quad \text{(any order you like)}
\]

Second, note that.

\[
\max(ab,ac) = a \max(b,c) \quad \text{(for } a \geq 0)\]

So, as you might expect

\[
\max_{x_i,x_j}(x_i,y_j) = \left( \max_{x_i}(x_i) \right) \left( \max_{y_j}(y_j) \right) \quad \text{(for } x_i,y_j \geq 0)\]

Recall inference on chains

\[
\begin{array}{cccccc}
  x_1 & x_2 & \ldots & x_{N-1} & x_N \\
\end{array}
\]

\[
p(x) = \psi_{1,2}(x_1,x_2) \psi_{2,3}(x_2,x_3) \ldots \psi_{N-2,N-1}(x_{N-2},x_{N-1}) \psi_{N-1,N}(x_{N-1},x_N)
\]

Naive computation of \( \arg \max_x p(x) \)

would evaluate the above for each value of \( x \), and take the max.

Too expensive!!

Recall speeding up marginalization

\[
p(x_i) = \left( \sum_{x_{i-1}} \sum_{x_{i+1}} \ldots \sum_{x_N} \prod_{j=1}^{N-1} \psi_{i,j}(x_{i-1},x_{i+1}) \right) \left( \sum_{x_{i+1}} \ldots \sum_{x_N} \prod_{j=i+2}^{N-1} \psi_{i,j}(x_i,x_{i+1}) \right)
\]

where

\[
\sum_{x_{i-1}} \ldots \sum_{x_{i+1}} \prod_{j=1}^{N-1} \psi_{i,j}(x_{i-1},x_{i+1}) = \left[ \sum_{x_{i-1}} \psi_{i-1,i}(x_{i-1},x_i) \right] \left[ \sum_{x_{i+1}} \psi_{i,i+1}(x_i,x_{i+1}) \right] \left[ \sum_{x_N} \psi_{N-1,N}(x_{N-1},x_N) \right] \ldots
\]

and

\[
\sum_{x_{i+1}} \ldots \sum_{x_N} \prod_{j=i+2}^{N-1} \psi_{i,j}(x_i,x_{i+1}) = \left[ \sum_{x_{i+1}} \psi_{i+1,i}(x_{i+1},x_i) \right] \left[ \sum_{x_N} \psi_{N-1,N}(x_N,x_{N+1}) \right] \left[ \sum_{x_{i-1}} \psi_{i-2,i+1}(x_{i-2},x_{i+1}) \right] \ldots
\]

What if we could do with \( \max() \) what we are doing with \( \sum \)?
Max on chains

\[ p(x) = \psi_{1,2}(x_1, x_2) \psi_{2,3}(x_2, x_3) \cdots \psi_{N-2, N-1}(x_{N-2}, x_{N-1}) \psi_{N-1, N}(x_{N-1}, x_N) \]

\[
\max_x p(x) = \max_x \left\{ \psi_{1,2}(x_1, x_2) \psi_{2,3}(x_2, x_3) \cdots \psi_{N-2, N-1}(x_{N-2}, x_{N-1}) \psi_{N-1, N}(x_{N-1}, x_N) \right\}
\]

\[
= \max_{x_1} \left[ \max_{x_2} \left[ \max_{x_3} \left[ \cdots \max_{x_{N-2}} \left( \max_{x_{N-1}} \left( \prod_{i=1}^{N-1} \psi_{i, i+1}(x_i, x_{i+1}) \right) \right) \right] \right] \right]
\]

(max products not involving \( x_n \) outside max \( \psi_{N-1, N}(x_{N-1}, x_N) \))

\[
= \max_{x_1} \left[ \max_{x_2} \left[ \max_{x_3} \left[ \cdots \max_{x_{N-2}} \left( \max_{x_{N-1}} \left( \prod_{i=1}^{N-1} \psi_{i, i+1}(x_i, x_{i+1}) \right) \right) \right] \right] \right]
\]

(continue moving factors out)

Max-sum algorithm

Two steps.
1) Compute the max while remembering certain computations
2) Compute a value of \( x \) that achieves the max

The message passing algorithm for step (1) is clear from the analog with the “sum-product” algorithm, except that it would then be called the “max-product” algorithm.

Computing long products loses precision (*), so we switch to log(), and call it the max-sum algorithm.

(*) Less of an issue with marginalization.

Max-sum algorithm (preliminaries)

Note that

\[
\ln \left( \max_x \left( p(x) \right) \right) = \max_x \left( \ln \left( p(x) \right) \right)
\]

And we have

\[
\ln \left( \max \left( ab, ac \right) \right) = \max \left( \ln(a) + \ln(b), \ln(a) + \ln(c) \right)
\]

\[
= \ln(a) + \max \left( \ln(b), \ln(c) \right)
\]

(In general, \( \max \left( x + y, x + z \right) = x + \max(y, z) \))

Can also get this from taking logs of product version, namely: \( \max(ab, ac) = a \cdot \max(b, c) \), for \( a \geq 0 \)

Max-sum algorithm

Develop using the analogy with the sum-product algorithm

Sums become max() and products become sums (over logs)

We will use the same notation for messages, but the semantics is a bit different (as above) and the quantities are always logs.
Max-sum algorithm

Develop using the analogy with the sum-product algorithm

\[ \mu_{f \rightarrow x}(x) = \max_{s_1, s_2, \ldots, s_M} \left[ \ln f(x, s_1, s_2, \ldots, s_M) + \sum_{\text{node}(f) \in t} \mu_{s \rightarrow f}(x) \right] \]

and

\[ \mu_{x \rightarrow f}(x) = \sum_{\text{root}(f) \in t} \mu_{f \rightarrow x}(x) \]

Recall that in sum-product:

\[ \mu_{f \rightarrow x}(x) = \sum_{s_1} \cdots \sum_{s_M} \ln f(x, s_1, s_2, \ldots, s_M) \prod_{\text{node}(f) \in t} \mu_{s \rightarrow f}(x) \]

Max-sum algorithm

Working now in analogy with the sum-product algorithm

For initialization at leaf nodes

\[ \mu_{f \rightarrow x}(x) = 0 \]

and

\[ \mu_{x \rightarrow f}(x) = \ln(f(x)) \]
Max-sum algorithm

Now we need to find an $x$ where $p()$ reaches the max.

This does not have an exact analogy in the sum-product algorithm.

**Why we do not know $x$ yet:**

The factor-to-node messages takes a distribution for the maxima over the upstream variables, and multiplies it by the factor (sum using logs), and reports a new distribution.

We do not yet know which value in the new distribution will be part of the maximum (it is not necessarily argmax of the reported distribution).

---

Max-sum algorithm (back-tracking)

In more detail, when we compute

$$
\mu_{f \rightarrow y}(x) = \max_{x_1, x_2, \ldots, x_M} \left[ \ln f(x, x_1, x_2, \ldots, x_M) + \sum_{\text{other nodes } \not= x} \mu_{x \rightarrow y}(x_n) \right]
$$

store

$$
\phi(x) = \arg \max_{x_1, x_2, \ldots, x_M} \left[ \ln f(x, x_1, x_2, \ldots, x_M) + \sum_{\text{other nodes } \not= x} \mu_{x \rightarrow y}(x_n) \right]
$$

This records the downstream choices for any upstream choice of $x$. $\phi(x)$ stored $M$ values for each value of $x$.

Then, once we know the overall max, we can recover a set of $x_i$ that leads to it by backtracking.
Max-sum in pictures

Two values give the same max. The root needs to choose one and sends its choice back towards the leaves.

Look up stored back pointer for the back-traced value. This back pointer, $\theta(x_1)$, has indices of the variables $x_1$ and $x_2$ that correspond to the chosen max.

The stored values for the argmax() for $x_3$ are now passed back to the source of $x_1$ and $x_2$.

Max over all variables except $x_3$. If there are duplicates (e.g., dark blue), then we choose one. Each chosen max (magenta dots) is associated with a back pointer for that slice: $\theta(x)$.

Product (sum in logs) of the incoming messages (e.g., $p(x_1)$ and $p(x_2)$) and the factor (e.g., $p(x_3|x_1, x_2)$).

The stored values for the argmax() for $x_2$ are now passed back to the source of $x_1$ and $x_2$.

Max-sum in pictures

Two values give the same max. The root needs to choose one and sends its choice back towards the leaves.

Look up stored back pointer for the back-traced value. This back pointer, $\theta(x_1)$, has indices of the variables $x_1$ and $x_2$ that correspond to the chosen max.

The stored values for the argmax() for $x_3$ are now passed back to the source of $x_1$ and $x_2$.

Max over all variables except $x_3$. If there are duplicates (e.g., dark blue), then we choose one. Each chosen max (magenta dots) is associated with a back pointer for that slice: $\theta(x)$.

Product (sum in logs) of the incoming messages (e.g., $p(x_1)$ and $p(x_2)$) and the factor (e.g., $p(x_3|x_1, x_2)$).