Lecture : Revisiting BP decoding from the lens of the sum-product algorithm + Some simplifications in implementation

In this lecture, we shall first take a brief detour from the world of LDPC codes and examine a seemingly different problem: that of computing marginals of a structured joint probability distribution.

[Examples and discussion heavily based on "Marginalization and Factor Graphs," lecture slides by Prof. Henry Pfister for the course "Advanced Channel Coding"]

Recall that the problem of decoding is intimately tied to manipulations of the posterior probability distribution:

Suppose that we have a random vector $(X_1, X_2, \ldots, X_6)$ that obeys

$$P_{\underline{X}}[X_1 = x_1, X_2 = x_2, \ldots, X_6 = x_6 \mid \underline{y}] \propto f(x_1, x_2, \ldots, x_6)$$
$$= f_1(x_1, x_2, x_3) f_2(x_1, x_4, x_6) \cdot f_3(x_4) f_4(x_4, x_5),$$

for some suitably defined functions $f, f_1 - f_4$.
Let S_i be the indices of the variables that f_i depends on, $i \in [4]$.

① Block-MAP (or simply, MAP) decoding: Compute

$$\underline{\hat{X}} = \underset{\underline{x}}{\operatorname{argmax}}\ f(x_1, \ldots, x_6) = \underset{\underline{x}}{\operatorname{argmax}} \prod_{i=1}^{4} f_i(x_{S_i})$$

② Bit-MAP (or symbol-MAP) decoding: Compute

$$\hat{x}_i = \underset{x_i \in \{0,1\}}{\arg\max} \sum_{x_{\sim i}} \prod_{i=1}^{4} f_i(x_{S_i})$$

↳ short for $x_{[6][i]}$.

As it turns out, the operations "maximum-of-product" and "sum-of-product" can be carried out efficiently via the so-called sum-product algorithm on graphs.

[To carry out a "maximum-of-product", simply replace "$\sum$" with "max" everywhere in what follows. The most general exposition of such algorithms can be found in [S.M. Aji and R.J. McEliece, "The generalized distributive law," IEEE T-IT (2000)].

An illustration of the sum-product algorithm

Consider the function f that admits a factorization as above. Note that computing, for each $x_1 \in \mathcal{X} = \{0,1\}$,

$$g(x_1) = \sum_{x_{\sim 1}} f(x_1, x_2, \ldots, x_6)$$

naively requires $|\mathcal{X}|^6$ operations.

However, via the factorization of f, we have

$$g(x_1) = \sum_{x_2^5} f_1(x_1, x_2, x_3)\, f_3(x_4)\, f_4(x_4, x_5) \left[\sum_{x_6} f_2(x_1, x_4, x_6)\right]$$

$$= \sum_{x_2^4} f_1(x_1, x_2, x_3)\, f_3(x_4) \left[\sum_{x_5} f_4(x_4, x_5)\right]\left[\sum_{x_6} f_2(x_1, x_4, x_6)\right]$$

can be carried out parallely

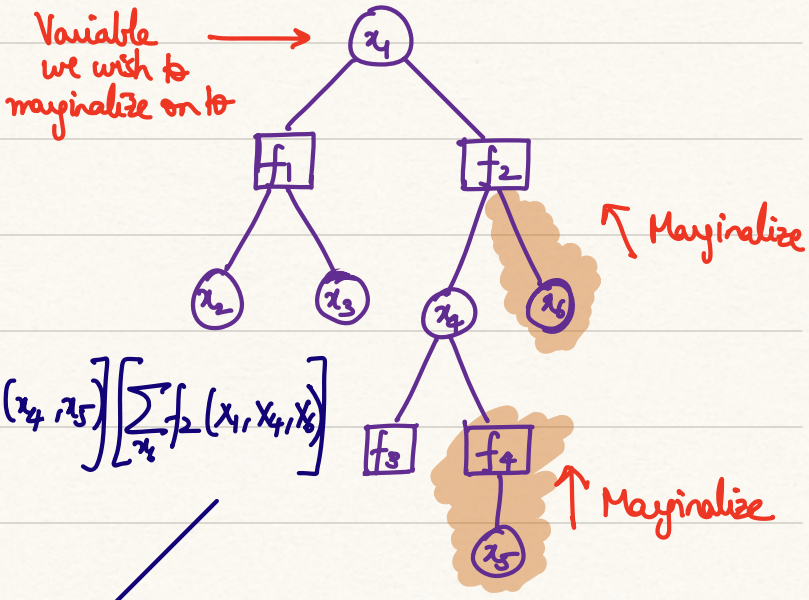
$$= \sum_{x_2^3} f_1(x_1, x_2, x_3) \left[\sum_{x_4} f_3(x_4) \left[\sum_{x_5} f_4(x_4, x_5)\right]\left[\sum_{x_6} f_2(x_1, x_4, x_6)\right]\right]$$

$$= \sum_{x_2} \left[\sum_{x_3} f_1(x_1, x_2, x_3)\right]\left[\sum_{x_4} f_3(x_4) \left[\sum_{x_5} f_4(x_4, x_5)\right]\left[\sum_{x_6} f_2(x_1, x_4, x_6)\right]\right].$$

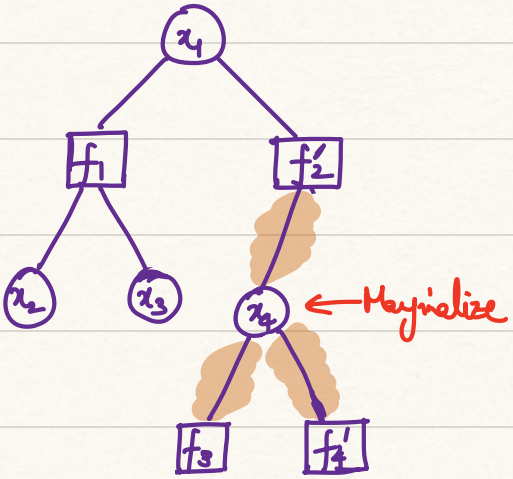
The final summation can be carried out with fewer operations [can we compute the # of operations in class?].

The above approach of "grouping summations together" can be visualized via a graphical model, the so-called factor graph [Recall the terminology of "factor nodes" in an earlier lecture].

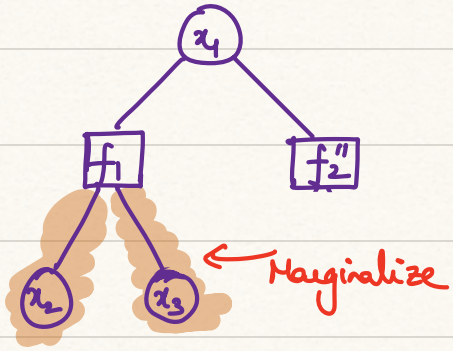
Factor graph representing f :



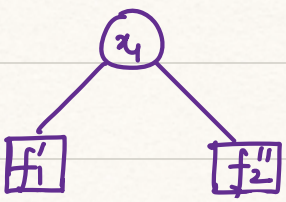
$$\left[\sum_{x_5} f_4(x_4, x_5)\right]\left[\sum_{x_6} f_2(x_1, x_4, x_6)\right]$$



$$\sum_{x_4} f_3(x_4)\, f_4'(x_4)\, f_2'(x_1, x_4)$$



$$\sum_{x_2, x_3} f_1(x_1, x_2, x_3)$$



$$f_1'(x_1) \cdot f_2''(x_1)$$

The steps carried out above are precisely the variable → factor and factor → variable message passing updates we had seen last class.

- Initialize: $\hat{\mu}_{i\to a}^{(1)}(x_i) = 1 \quad , \forall\ (i,a)$

- (i) Factor → variable:

$$\hat{\nu}_{a\to i}^{(t)}(x_i) = \sum_{x_{N(a)\setminus\{i\}}} f_a(x_a) \cdot \prod_{j\in N(a)\setminus\{i\}} \hat{\mu}_{j\to a}(x_j)$$

(ii) Variable → Factor:

$$\hat{\mu}_{i\to a}^{(t+1)}(x_i) = \prod_{b\in N(i)\setminus\{a\}} \hat{\nu}_{b\to i}^{(t)}(x_i)$$

After convergence, output the product of messages at the root.

[Note that the order of steps (i) & (ii) above is reversed compared to that in last lecture; this order does not really matter...]

FACT: (i) If the factor graph is a tree (connected graph with no cycles), then the message-passing algorithm converges (all messages converge) for $t \geq t^* = \left\lceil \frac{\text{diam(factor graph)}}{2} \right\rceil$.

(ii) In practice, repeating the steps above can result in numerical underflow. To alleviate this, scale messages to sum to 1. This will result in the final output "marginals" to also sum to 1.

Simplifying the message-passing algorithm for graph-based codes

We now discuss a clever technique for implementing BP decoding of

graph-based (say, LDPC) binary codes.

Define $L_{i\to a}^{(t)} \triangleq \ln \frac{\mu_{i\to a}^{(t)}(0)}{\mu_{i\to a}^{(t)}(1)}$ and $L_{a\to i}^{(t)} \triangleq \ln \frac{\nu_{a\to i}^{(t)}(0)}{\nu_{a\to i}^{(t)}(1)}$.

messages are now scalars!

Then,

- Step 1(a) [see prev. lecture] reduces to

$$L_{i\to a}^{(t)} = \sum_{b\in N(i)\setminus\{a\}} L_{a\to i}^{(t)}$$

- Step 1(b) [see prev. lecture] reduces to

$$L_{a\to i}^{(t+1)} = 2\tanh^{-1}\left(\prod_{j\in N(a)\setminus\{i\}} \tanh\left(\frac{1}{2} L_{j\to a}^{(t)}\right)\right)$$

[Looks fancy!]

- Step 3 [see prev lecture] reduces to simply computing the sum of incoming messages (scalars) at each bit i and declaring $\hat{c}_i$ to be 0 if this sum ≥ 0 and 1, otherwise.

Programming Exercise: Implement the above form of the BP decoder for an LDPC code from the Gallager ensemble, from which the all-zeros codeword is transmitted over the BSC(p), for varying p.

* Can be repeated for the BI-AWGN channel.