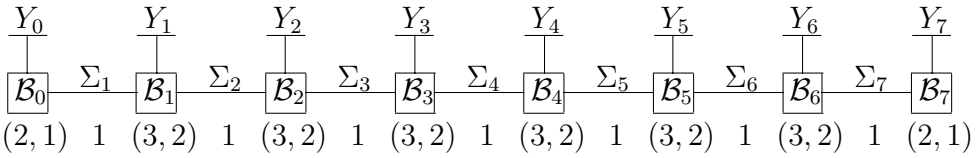
Problem Set 9 Solutions

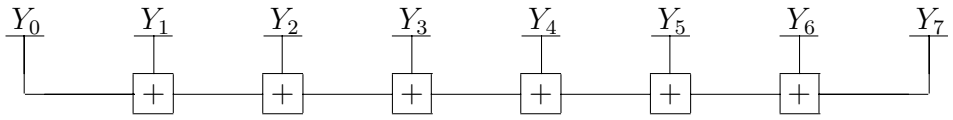
Problem 8.3 (revised) (BCJR (sum-product) decoding of SPC codes)

As shown in Problem 6.4 or Figure 1 of Chapter 10, any $(n, n-1, 2)$ *single-parity-check code has a two-state trellis diagram. Consider the* $(8, 7, 2)$ *code, and let the received sequence from a discrete-time AWGN channel with input alphabet* $\{\pm 1\}$ *and noise variance* $\sigma^2 = 1$ *be given by* $\mathbf{r} = (0.1, -1.0, -0.7, 0.8, 1.1, 0.3, -0.9, 0.5)$. *Perform BCJR (sum-product) decoding of this sequence to determine the APP vector for each symbol* Y_k. *[Hint: the special algorithm given in Section 13.5.2 (see Problem 9.5) to implement the sum-product update rule for zero-sum nodes may be helpful here.]*

First, let us draw the normal graph of a minimal trellis realization of the $(8, 7, 2)$ code. This graph is shown abstractly below:



Moreover, it is easy to see that the $(3, 2)$ branch constraint codes are all $(3, 2, 2)$ zero-sum codes, and the $(2, 1)$ codes are simple repetition codes that need not actually be shown. Therefore the trellis realization may be drawn simply as follows:



Note that this trellis realization of the $(8, 7, 2)$ code is another cycle-free realization that uses 6 $(3, 2, 2)$ zero-sum constraints, as in the reduced HT realization of Problem 8.1; however, the diameter of this realization is 5 (which is as large as it could possibly be).

For a binary-input Gaussian-noise channel with inputs $\{\pm 1\}$ and Gaussian conditional probability density $p(r \mid y) = (2\pi\sigma)^{-1/2} \exp -(r-y)^2/2\sigma^2$, the *a posteriori* probability (APP) of $y \in \{\pm 1\}$ given a received value r is, by Bayes' rule,

$$p(y \mid r) = \frac{p(r \mid y)}{p(r \mid 1) + p(r \mid -1)} = \frac{e^{ry/\sigma^2}}{e^{r/\sigma^2} + e^{-r/\sigma^2}}.$$

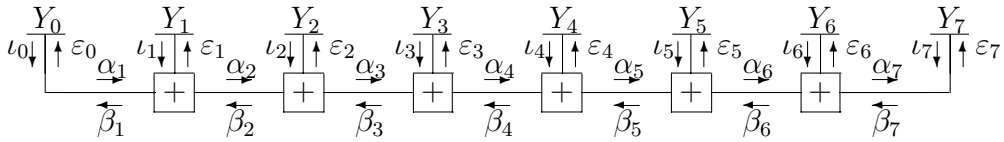
Here $\sigma^2 = 1$ (so SNR = 1 (0 dB); *i.e.,* the channel is very noisy).

The two values $p(\pm 1 \mid r_k)$ form the "intrinsic information" message $\iota_k = \{p_{0k}, p_{1k}\}$ derived from each received symbol $r_k, 0 \le k \le 7$. These values are computed from the received vector $\mathbf{r} = (0.1, -1.0, -0.7, 0.8, 1.1, 0.3, -0.9, 0.5)$ in the following table. (Note that it would have sufficed to use the unnormalized pair $\{e^{r_k/\sigma^2}, e^{-r_k/\sigma^2}\}$.) We also compute the Hadamard transform $\{p_{0k} + p_{1k} = 1, p_{0k} - p_{1k} = I_k\}$ of each pair of values for later use.

r_k	p_{0k}	p_{1k}	I_k	
$r_0 = +0.1$	0.55	0.45	1	$+0.10$
$r_1 = -1.0$	0.12	0.88	1	-0.76
$r_2 = -0.7$	0.20	0.80	1	-0.60
$r_3 = +0.8$	0.83	0.17	1	$+0.66$
$r_4 = +1.1$	0.90	0.10	1	$+0.80$
$r_5 = +0.3$	0.65	0.35	1	$+0.30$
$r_6 = -0.9$	0.14	0.86	1	-0.72
$r_7 = +0.5$	0.73	0.37	1	$+0.46$

Note that $I_k = \tanh r_k/\sigma^2$, so $I_k \approx r_k/\sigma^2$ for small r_k. The sign of I_k represents a "hard decision," whereas the magnitude $0 \le |I_k| \le 1$ represents the reliability of that decision.

We then propagate the APP messages through the graph below, using the BCJR (sum-product) algorithm. Note that in the forward direction $\alpha_1 = \iota_0$, α_k is the sum-product update of α_{k-1} and ι_{k-1} for $2 \le k \le 7$, and finally $\varepsilon_7 = \alpha_7$. Similarly, in the backward direction, $\beta_7 = \iota_7$, β_k is the sum-product update of β_{k+1} and ι_k for $6 \ge k \ge 1$, and finally $\varepsilon_0 = \beta_1$. Then ε_k is the sum-product update of α_k and β_{k+1} for $1 \le k \le 6$.



For a $(3, 2, 2)$ constraint code $\mathcal{C}_k$, the set of past 2-tuples consistent with a 0 value for any incident variable is $\mathcal{C}_k(0) = \{00, 11\}$, and similarly $\mathcal{C}_k(1) = \{01, 10\}$. Therefore the sum-product update rule is

$$\begin{aligned} p(0 \mid \mathbf{r}_{|\mathcal{P}}) &= p(0 \mid \mathbf{r}_{|\mathcal{P}_{j'}})p(0 \mid \mathbf{r}_{|\mathcal{P}_{j''}}) + p(1 \mid \mathbf{r}_{|\mathcal{P}_{j'}})p(1 \mid \mathbf{r}_{|\mathcal{P}_{j''}}); \\ p(1 \mid \mathbf{r}_{|\mathcal{P}}) &= p(0 \mid \mathbf{r}_{|\mathcal{P}_{j'}})p(1 \mid \mathbf{r}_{|\mathcal{P}_{j''}}) + p(0 \mid \mathbf{r}_{|\mathcal{P}_{j'}})p(1 \mid \mathbf{r}_{|\mathcal{P}_{j''}}), \end{aligned}$$

where $\mathcal{P}_{j'}$ and $\mathcal{P}_{j''}$ are the two "pasts" upstream of $\mathcal{P}$.

Alternatively, following the hint, we may use the special rule for zero-sum nodes to obtain the Hadamard transform of $(p(0 \mid \mathbf{r}_{|\mathcal{P}}), p(1 \mid \mathbf{r}_{|\mathcal{P}}))$ simply by componentwise multiplication of the Hadamard transforms of $(p(0 \mid \mathbf{r}_{|\mathcal{P}_{j'}}), p(1 \mid \mathbf{r}_{|\mathcal{P}_{j'}}))$ and $(p(0 \mid \mathbf{r}_{|\mathcal{P}_{j''}}), p(1 \mid \mathbf{r}_{|\mathcal{P}_{j''}}))$.

By either method, we obtain the following values for the forward messages $\alpha_k = \{\alpha_{0k}, \alpha_{1k}\}$ and their Hadamard transforms $\{\alpha_{0k} + \alpha_{1k} = 1, \alpha_{0k} - \alpha_{1k} = A_k\}$, the backward messages $\beta_k = \{\beta_{0k}, \beta_{1k}\}$ and their Hadamard transforms $\{1, B_k\}$, and the extrinsic information messages $\varepsilon_k = \{\varepsilon_{0k}, \varepsilon_{1k}\}$ and their Hadamard transforms $\{1, E_k\}$.

α_k	α_{0k}	α_{1k}	A_k	
$\alpha_1 = \iota_0$	0.55	0.45	1	$+0.10$
α_2	0.46	0.54	1	-0.08
α_3	0.525	0.475	1	$+0.05$
α_4	0.515	0.485	1	$+0.03$
α_5	0.51	0.49	1	$+0.02$
α_6	0.5035	0.4965	1	$+0.007$
α_7	0.4975	0.5025	1	-0.005

β_k	β_{0k}	β_{1k}	B_k	
$\beta_7 = \iota_7$	0.73	0.27	1	$+0.46$
β_6	0.335	0.665	1	-0.33
β_5	0.45	0.55	1	-0.10
β_4	0.46	0.54	1	-0.08
β_3	0.475	0.525	1	-0.05
β_2	0.515	0.485	1	$+0.03$
β_1	0.49	0.51	1	-0.02

ε_k	ε_{0k}	ε_{1k}	E_k	
$\varepsilon_0 = \beta_1$	0.49	0.51	1	-0.02
ε_1	0.5015	0.4985	1	$+0.003$
ε_2	0.5015	0.4985	1	$+0.003$
ε_3	0.498	0.502	1	-0.004
ε_4	0.4985	0.5015	1	-0.003
ε_5	0.4965	0.5035	1	-0.007
ε_6	0.5015	0.4985	1	$+0.003$
$\varepsilon_7 = \alpha_7$	0.4975	0.5025	1	-0.005

Notice how the reliability of the forward and backward APP messages α_k and β_k degenerates as more and more intrinsic information messages ι_k are incorporated into them.

The APP vectors $\{p(Y_k = 0 \mid \mathbf{r}), p(Y_k = 1 \mid \mathbf{r})\}$ for the symbol variables Y_k are ultimately obtained by componentwise multiplication of ι_k and ε_k, normalized. We note that for all k, since $\varepsilon_{0k} \approx \varepsilon_{1k} \approx \frac{1}{2}$, we have $\{p(Y_k = 0 \mid \mathbf{r}), p(Y_k = 1 \mid \mathbf{r})\} \approx \{\iota_{0k}, \iota_{1k}\}$; *i.e.,* the intrinsic information ι_k dominates. Thus if hard decisions are made on each symbol, the result is the same as if hard decisions had been made symbol-by-symbol based solely on the channel outputs r_k, and the resulting sequence of hard decisions is not a code sequence in the $(8, 7, 2)$ code.

In contrast, suppose that we perform the max-product (equivalent to the min-sum) algorithm with this received sequence. We obtain the following values:

α_k	α_{0k}	α_{1k}
$\alpha_1 = \iota_0$	0.55	0.45
α_2	0.45	0.55
α_3	0.55	0.45
α_4	0.55	0.45
α_5	0.55	0.45
α_6	0.55	0.45
α_7	0.45	0.55

β_k	β_{0k}	β_{1k}
$\beta_7 = \iota_7$	0.73	0.27
β_6	0.27	0.73
β_5	0.35	0.65
β_4	0.35	0.65
β_3	0.35	0.65
β_2	0.65	0.35
β_1	0.35	0.65

ε_k	ε_{0k}	ε_{1k}
$\varepsilon_0 = \beta_1$	0.35	0.65
ε_1	0.55	0.45
ε_2	0.55	0.45
ε_3	0.45	0.55
ε_4	0.45	0.55
ε_5	0.45	0.55
ε_6	0.55	0.45
$\varepsilon_7 = \alpha_7$	0.45	0.55

In this case, the reliability of a forward or backward message is the minimum reliability of any intrinsic information that is incorporated into it. Eventually, this means that the extrinsic information ε_0 dominates the intrinsic information ι_0 for the least reliable symbol Y_0, so the original "hard decision" is "corrected" in this case. The same "correction" would be performed by the Viterbi algorithm or by Wagner decoding.

The max-product (min-sum) algorithm finds the maximum-likelihood code sequence, whereas the BCJR (sum-product) algorithm computes the APP vector of each bit. A bit decision based on the maximum APP minimizes the bit error probability, so the bit error probability could be (slightly) lower with the BCJR algorithm. ML sequence detection minimizes the probability of decoding to the wrong codeword. The sequence of maximum-APP bits from the BCJR algorithm may not be a codeword, as we have just seen.

Compare the complexity of BCJR decoding to Viterbi algorithm decoding (Problem 6.6).

The BCJR algorithm is considerably more complex. For each trellis segment, it has to compute the sum-product update rule three times. The straightforward sum-product update rule involves four multiplications and two additions; the simplified rule requires a single multiplication. By contrast, the VA requires four additions and two comparisons per trellis segment. Computation of the intrinsic APP vector is also more complex, and requires estimation of the noise variance σ^2, which could be a problem. Finally, the logic of the BCJR algorithm is more complex, since it is two-way rather than one-way.

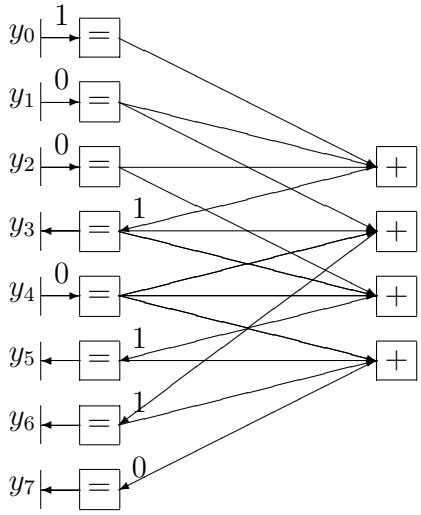
In short, the VA is significantly simpler than the BCJR algorithm. For this reason, the VA was preferred for trellis decoding for many years. The BCJR algorithm was resurrected with the advent of turbo codes, where the fact that it produces "soft" (APP) outputs is essential to its role as a part of iterative decoding algorithms.

(For this scenario, Wagner decoding is even simpler; see the solution to Problem 6.6.)

Problem 9.1 (Iterative decoding on the BEC)

(a) Using a graph of the $(8, 4, 4)$ *code like that of Figure 1 of Chapter 13 for iterative decoding, decode the received sequence* $(1, 0, 0, ?, 0, ?, ?, ?)$.

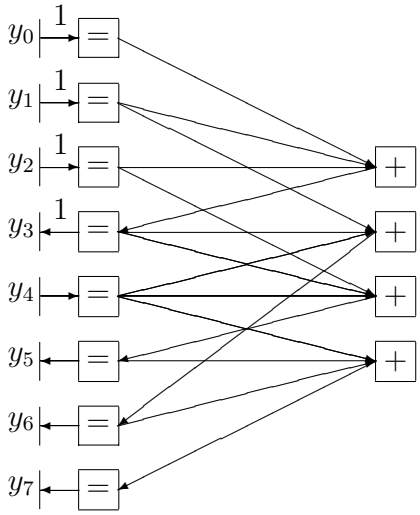
We may use the equivalent normal graph of Figure 3(b) of Chapter 11, which may be converted to a directed normal graph (encoder) as shown in Figure 15 of Chapter 11, repeated below.



The directed input edges show that (y_0, y_1, y_2, y_4) is an information set for this code. Given $(y_0, y_1, y_2, y_4) = (1, 0, 0, 0)$, the sum-product algorithm fills in the erasures by following the directed state edges as shown above. On the first iteration, the first check yields $y_3 = 1$; on the second iteration, the second and third checks yield $y_5 = y_6 = 1$; and on the third and last iteration, the last check yields $y_7 = 0$.

Then try to decode the received sequence $(1, 1, 1, 1, ?, ?, ?, ?)$. *Why does decoding fail in the latter case? Give both a local answer (based on the graph) and a global answer (based on the code).*

As shown in the graph below, iterative decoding stalls because at the initial iteration, the first check checks, and each of the last three checks has two or more erased inputs, so none of the checks yields any new symbol values. The symbols (y_4, y_5, y_6, y_7) and the last three checks are called a *stopping set*, because at least two of the symbols are involved in each of the three checks; therefore if all are erased, no progress can be made.

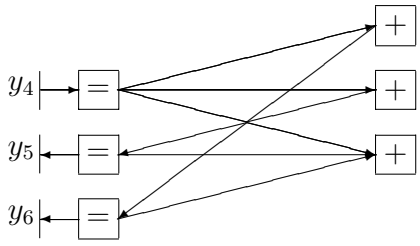


In this case, however, even a global (ML) decoder must fail, because there are two codewords with $(y_0, y_1, y_2, y_3) = (1, 1, 1, 1)$, namely 11110000 and 11111111.

(b) For the received sequence $(1, 1, 1, 1, ?, ?, ?, 0)$*, show that iterative decoding fails but that global (i.e., ML) decoding succeeds.*

In this case, there is only one codeword with $(y_0, y_1, y_2, y_3, y_7) = (1, 1, 1, 1, 0)$, namely 11110000, so global (ML) decoding will succeed.

On the other hand, iterative decoding will fail, because the symbols (y_4, y_5, y_6) and the last three checks form a stopping set of this graph:



Problem 9.2 (Simulation of LDPC decoding on a BEC)

(a) Perform a simulation of iterative decoding of a regular $(d_\lambda = 3, d_\rho = 6)$ *LDPC code on a BEC with* $p = 0.45$ *(i.e., on Figure 9 of Chapter 13), and show how decoding gets stuck at the first fixed point* $(q_{\ell \to r} \approx 0.35, q_{r \to \ell} \approx 0.89)$*. About how many iterations does it take to get stuck?*

We start with $q_{r \to \ell} = 1$ and use the sum-product update relationships developed for this case in Chapter 13:

$$q_{\ell \to r} = (0.45)(q_{r \to \ell})^2; \qquad q_{r \to \ell} = 1 - (1 - q_{\ell \to r})^5.$$

We obtain the following sequence:

$q_{r\to\ell}$	$q_{\ell\to r}$
1	0.45
0.949672	0.405844
0.925954	0.385826
0.912611	0.374786
0.904469	0.368129
0.899274	0.363912
0.895868	0.361161
0.893596	0.359331
0.892064	0.358100
0.891022	0.357264
0.890311	0.356694
0.889824	0.356304
0.889490	0.356036
0.889259	0.355852
0.889101	0.355725
0.888992	0.355638
0.888916	0.355577
0.888864	0.355536
0.888828	0.355507
0.888804	0.355487
0.888787	0.355474
0.888775	0.355464
0.888767	0.355458
0.888761	0.355453
0.888757	0.355450
0.888755	0.355448
0.888753	0.355447
0.888751	0.355446
0.888751	0.355445
0.888750	0.355444
0.888750	0.355444
0.888749	0.355444
0.888749	0.355444
...	...

The point is that it takes a very finite number of iterations to get stuck, about 15–30 full iterations, depending on how we define "stuck."

(b) By simulation of iterative decoding, compute the coordinates of the fixed point to six significant digits.

By the above simulation, the fixed point to six significant digits is at $(q_{r\to\ell}, q_{\ell\to r}) = (0.888749, 0.355444)$.

Problem 9.3 (Iterative decoding threshold)

By analysis or simulation, show that the smallest p such that the equation $x = 1-(1-px^2)^5$ has a solution in the interval $0 < x < 1$ is $p^* = 0.429....$ Explain the significance of this calculation for iterative decoding of LDPC codes on a BEC.

A simulation such as that in Problem 9.2 will succeed for $p = 0.429$ and get stuck for $p = 0.430$. Therefore p^* is somewhere between these two values.

The significance of this calculation is that a rate-1/2 regular $(d_\lambda = 3, d_\rho = 6)$ LDPC code with iterative decoding can be used on any BEC with $p < p^* = 0.429....$ This is not as good as a random linear rate-1/2 code with an ML decoder, which should be able to be used on any BEC with $p < 0.5$, but it is pretty good.

It has now been shown that there exist irregular rate-1/2 LDPC codes that can be used with iterative decoding on any BEC with $p < 0.5$.

Problem 9.4 (Stability condition)

(a) Show that if the minimum left degree of an irregular LDPC code is 3, then the stability condition necessarily holds.

If $\lambda(x) = \sum_d \lambda_d x^{d-1} = \lambda_3 x^2 + \cdots$, then $\lambda'(0) = 0$, so the stability condition $p\lambda'(0)\rho'(1) < 1$ necessarily holds.

(b) Argue that such a left degree distribution $\lambda(x)$ cannot be capacity-approaching, in view of Theorem 13.2.

If $\lambda(x) = \lambda_3 x^2 + \cdots$, then in the neighborhood of the top right point $(0, 0)$ of the EXIT chart, the curve $q_{\ell\to r} = p\lambda(q_{r\to\ell})$ is approximately quadratic, $q_{\ell\to r} \approx p\lambda_3 (q_{r\to\ell})^2$, whereas the curve $q_{r\to\ell} = 1-\rho(1-q_{\ell\to r})$ is approximately linear, $q_{r\to\ell} \approx \rho'(1) q_{\ell\to r}$. Therefore there must be a gap of nonnegligible area between these two curves in the neighborhood of the $(0, 0)$ point. By the area theorem, if the gap between the two curves has a nonnegligible area, then capacity cannot be approached arbitrarily closely.

Problem 9.5 (Sum-product update rule for zero-sum nodes)

(a) Prove that the algorithm of Section 13.5.2 implements the sum-product update rule for a zero-sum node, up to scale. [Hint: observe that in the product $\prod_j (w_0^{\text{in},j} - w_1^{\text{in},j})$, the terms with positive signs sum to w_0^{out}, whereas the terms with negative signs sum to w_1^{out}.]

Following the hint, we have

$$W_0^{\text{out}} = \prod_j W_0^{\text{in},j} = \prod_j (w_0^{\text{in},j} + w_1^{\text{in},j}); \qquad W_1^{\text{out}} = \prod_j W_1^{\text{in},j} = \prod_j (w_0^{\text{in},j} - w_1^{\text{in},j}).$$

In the latter equation, we observe that the terms with positive signs sum to w_0^{out}, whereas the terms with negative signs sum to w_1^{out}, so the second product equals $w_0^{\text{out}} - w_1^{\text{out}}$. The same terms occur in the former equation, but all with positive signs, so the first product equals $w_0^{\text{out}} + w_1^{\text{out}}$. Therefore the pair $(W_0^{\text{out}}, W_1^{\text{out}})$ is the Hadamard transform of $(w_0^{\text{out}}, w_1^{\text{out}})$, so we may obtain $(w_0^{\text{out}}, w_1^{\text{out}})$ from $(W_0^{\text{out}}, W_1^{\text{out}})$ by taking the inverse Hadamard transform (up to scale): $w_0^{\text{out}} = W_0^{\text{out}} + W_1^{\text{out}}$; $w_1^{\text{out}} = W_0^{\text{out}} - W_1^{\text{out}}$.

(b) Show that if we interchange $w_0^{\text{in},j}$ and $w_1^{\text{in},j}$ in an even number of incoming APP vectors, then the outgoing APP vector $\{w_0^{\text{out}}, w_1^{\text{out}}\}$ is unchanged. On the other hand, show that if we interchange $w_0^{\text{in},j}$ and $w_1^{\text{in},j}$ in an odd number of incoming APP vectors, then the components w_0^{out} and w_1^{out} of the outgoing APP vector are interchanged.

Interchanging $w_0^{\text{in},j}$ and $w_1^{\text{in},j}$ changes the sign of $W_1^{\text{in},j}$, while leaving $W_0^{\text{in},j}$ unchanged. Therefore if there an even number of such interchanges, W_0^{out} and W_1^{out} are unchanged, and so therefore are w_0^{out} and w_1^{out}. If there an odd number of such interchanges, then the sign of W_1^{out} is changed, which results in an interchange between w_0^{out} and w_1^{out}.

(c) Show that if we replace APP weight vectors $\{w_0, w_1\}$ by log likelihood ratios $\Lambda = \ln w_0/w_1$, then the zero-sum sum-product update rule reduces to the "tanh rule"

$$\Lambda^{\text{out}} = \ln\left(\frac{1+\prod_j \tanh \Lambda^{\text{in},j}/2}{1-\prod_j \tanh \Lambda^{\text{in},j}/2}\right),$$

where the hyperbolic tangent is defined by $\tanh x = (e^x - e^{-x})/(e^x + e^{-x})$.

Since $\Lambda^{\text{in},j}/2 = \ln(w_0^{\text{in},j}/w_1^{\text{in},j})^{1/2}$, we have

$$\tanh \Lambda^{\text{in},j}/2 = \frac{(w_0^{\text{in},j}/w_1^{\text{in},j})^{1/2} - (w_0^{\text{in},j}/w_1^{\text{in},j})^{-1/2}}{(w_0^{\text{in},j}/w_1^{\text{in},j})^{1/2} + (w_0^{\text{in},j}/w_1^{\text{in},j})^{-1/2}} = \frac{w_0^{\text{in},j} - w_1^{\text{in},j}}{w_0^{\text{in},j} + w_1^{\text{in},j}} = \frac{W_1^{\text{in},j}}{W_0^{\text{in},j}}.$$

(Note that if $(w_0^{\text{in},j}, w_1^{\text{in},j})$ are normalized so that $w_0^{\text{in},j} + w_1^{\text{in},j} = W_0^{\text{in},j} = 1$, then $\tanh \Lambda^{\text{in},j}/2 = W_1^{\text{in},j}$.) Consequently

$$\prod_j \tanh \Lambda^{\text{in},j}/2 = \prod_j \frac{W_1^{\text{in},j}}{W_0^{\text{in},j}} = \frac{W_1^{\text{out}}}{W_0^{\text{out}}}.$$

It follows that

$$\ln\left(\frac{1+\prod_j \tanh \Lambda^{\text{in},j}/2}{1-\prod_j \tanh \Lambda^{\text{in},j}/2}\right) = \ln \frac{W_0^{\text{out}} + W_1^{\text{out}}}{W_0^{\text{out}} - W_1^{\text{out}}} = \ln \frac{w_0^{\text{out}}}{w_1^{\text{out}}} = \Lambda^{\text{out}}.$$

(d) Show that the "tanh rule" may alternatively be written as

$$\tanh \Lambda^{\text{out}}/2 = \prod_j \tanh \Lambda^{\text{in},j}/2.$$

As with the expression for $\tanh \Lambda^{\text{in},j}/2$ above, we have

$$\tanh \Lambda^{\text{out}}/2 = \frac{W_1^{\text{out}}}{W_0^{\text{out}}},$$

which we have shown above to be equal to $\prod_j \tanh \Lambda^{\text{in},j}/2$.