Information Theory

http://www.inf.ed.ac.uk/teaching/courses/it/

Week 8

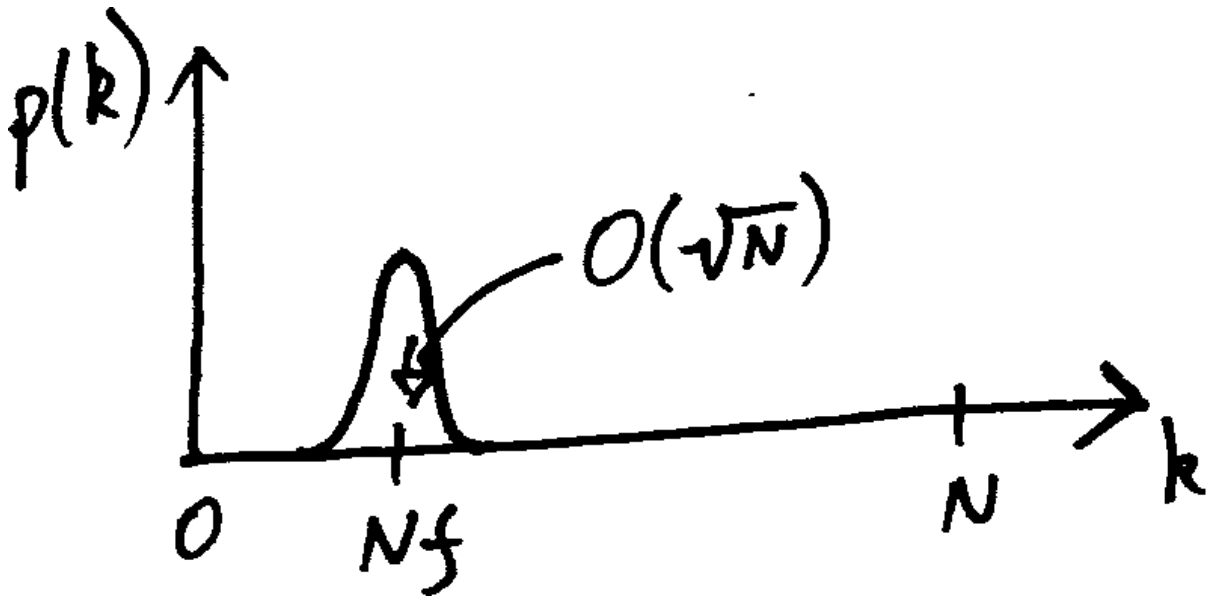
Noisy channel coding theorem and LDPC codes

Iain Murray, 2012

School of Informatics, University of Edinburgh

Typical sets revisited

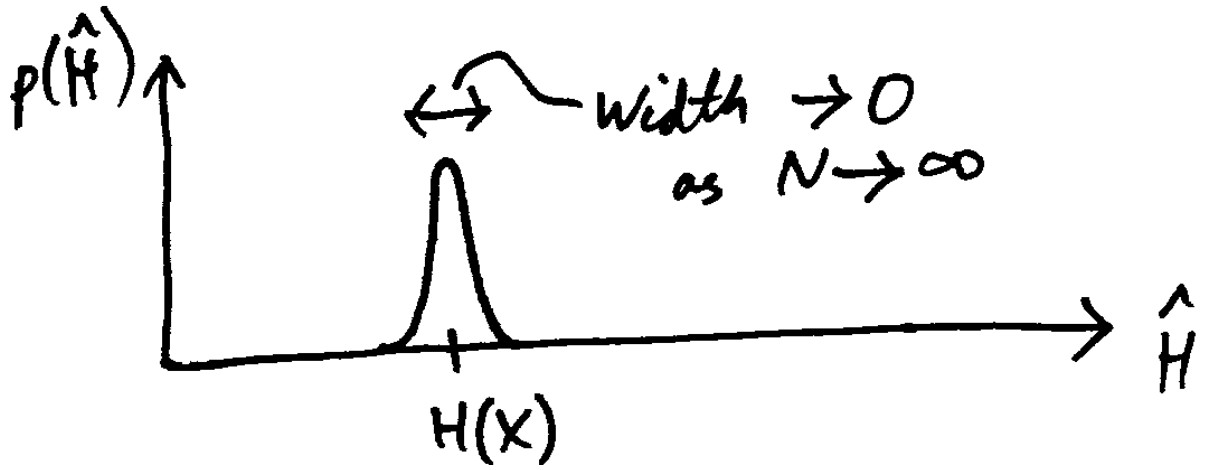
Week 2: looked at $k = \sum_i x_i, \;\; x_i \sim \text{Bernoulli}(f)$



Saw number of 1's is almost always in narrow range around expected number. Indexing this 'typical set' was the cost of compression.

Typical sets: general alphabets

More generally look at $\hat{H} = \frac{1}{N}\sum_i \log \frac{1}{P(x_i)}, \;\; x_i \sim P$



Define typical set: $\mathbf{x} \in T_{N,\beta}$ if $\left|\frac{1}{N}\log\frac{1}{P(\mathbf{x})} - H(X)\right| < \beta$

For any β, $P(\mathbf{x} \in T_{N,\beta}) > 1-\delta$, for any δ if N big enough

See MacKay, Ch. 4

Source Coding Theorem

(MacKay, p82–3 for details)

Min probability in $T_{N,\beta}$ is $2^{-N(H(X)+\beta)}$

Therefore typical set has size $\leq 2^{N(H(X)+\beta)}$

For large N can set β small

Index almost all strings with $\log_2 2^{NH(X)} = NH(X)$ bits

We now extend ideas of typical sets to joint ensembles of inputs and outputs of noisy channels. . .

Jointly typical sequences

For $n = 1 \ldots N$: $x_n \sim \mathbf{p}_X$

Send $\mathbf{x}$ over extended channel: $y_n \sim Q_{\cdot|x_n}$

Jointly typical:

$(\mathbf{x}, \mathbf{y}) \in J_{N,\beta}$ if $\left| \frac{1}{N} \log \frac{1}{P(\mathbf{x},\mathbf{y})} - H(X,Y) \right| < \beta$

There are $\leq 2^{N(H(X,Y)+\beta)}$ jointly typical sequences

Chance of being jointly typical

$(\mathbf{x}, \mathbf{y})$ from channel are jointly typical with prob $1-\delta$

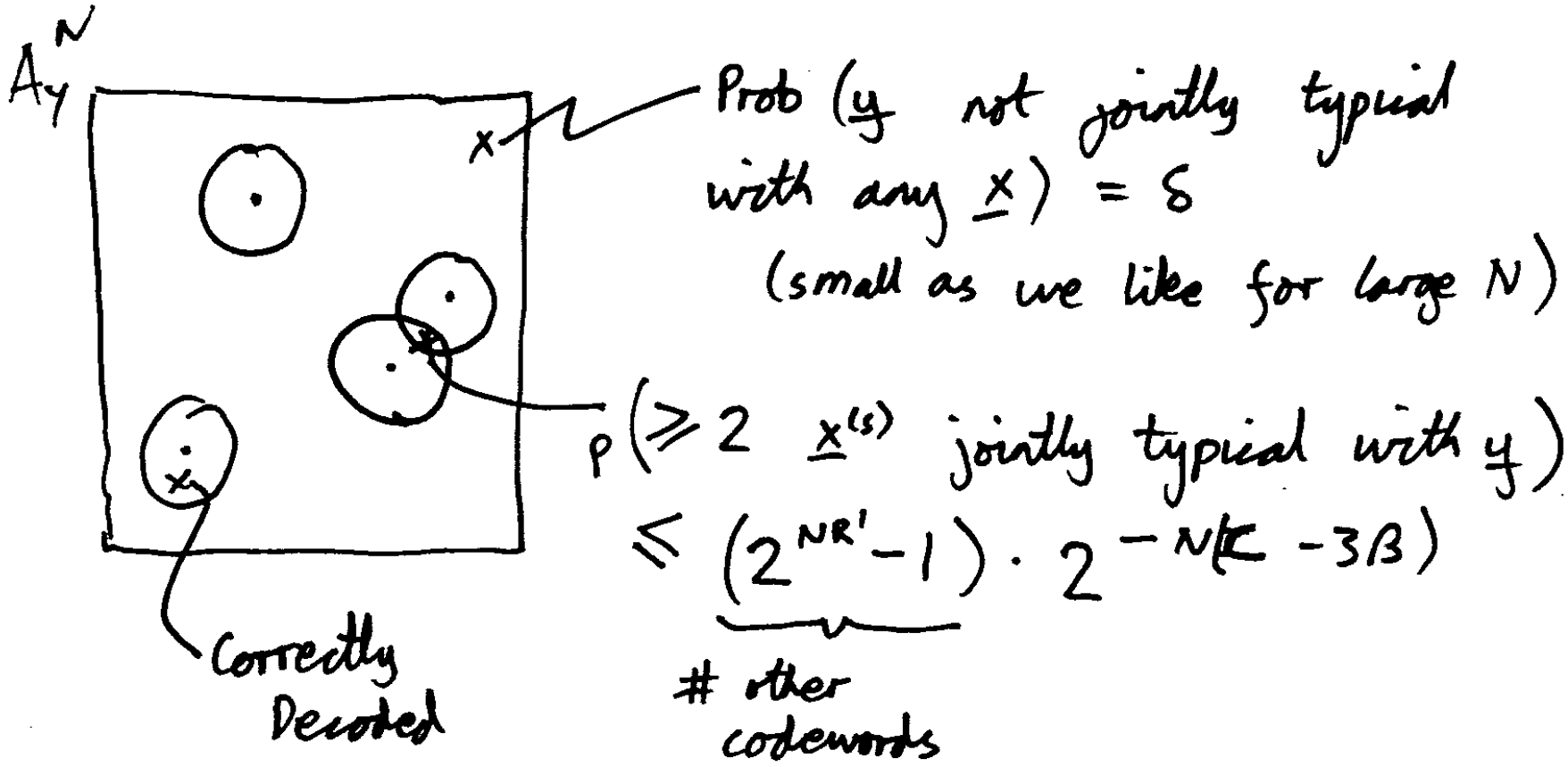
$(\mathbf{x}', \mathbf{y}')$ generated independently are rarely jointly typical

$$
\begin{aligned}
P(\mathbf{x}', \mathbf{y}' \in J_{N,\beta}) &= \sum_{(\mathbf{x},\mathbf{y}) \in J_{N,\beta}} P(\mathbf{x})\, P(\mathbf{y}) \\
&\leq |J_{N,\beta}|\, 2^{-N(H(X)-\beta)}\, 2^{-N(H(Y)-\beta)} \\
&\leq 2^{N(H(X,Y)-H(X)-H(Y)+3\beta)} \\
&\leq 2^{-N(I(X;Y)-3\beta)} \\
&\leq 2^{-N(C-3\beta)}, \quad \text{for optimal } \mathbf{p}_X
\end{aligned}
$$

Random typical set code

Randomly choose $S = 2^{NR'}$ codewords $\{\mathbf{x}^{(s)}\}$

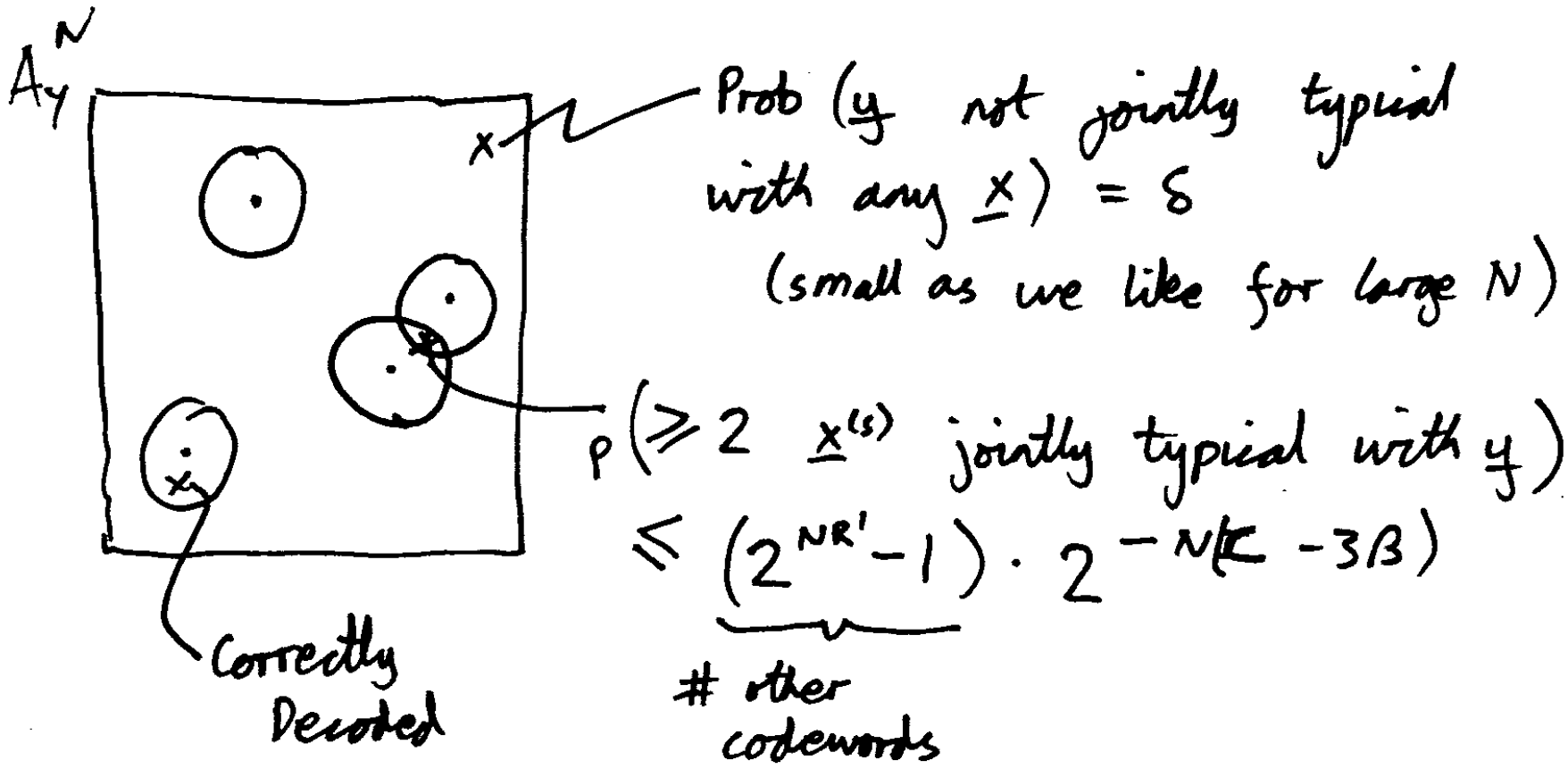
Decode $\mathbf{y} \to \hat{s}$ if $(\mathbf{y}, \mathbf{x}^{(\hat{s})}) \in J_{N,\beta}$
and no other $(\mathbf{y}, \mathbf{x}^{(s')}) \in J_{N,\beta}$



Error rate averaged over codes

Set rate $R' < C - 3\beta$. For large N prob. confusion $< \delta$

Total error probability on average $< 2\delta$



Error for a particular code

We randomly drew all the codewords for each symbol sent.

Block error rate averaged over all codes:

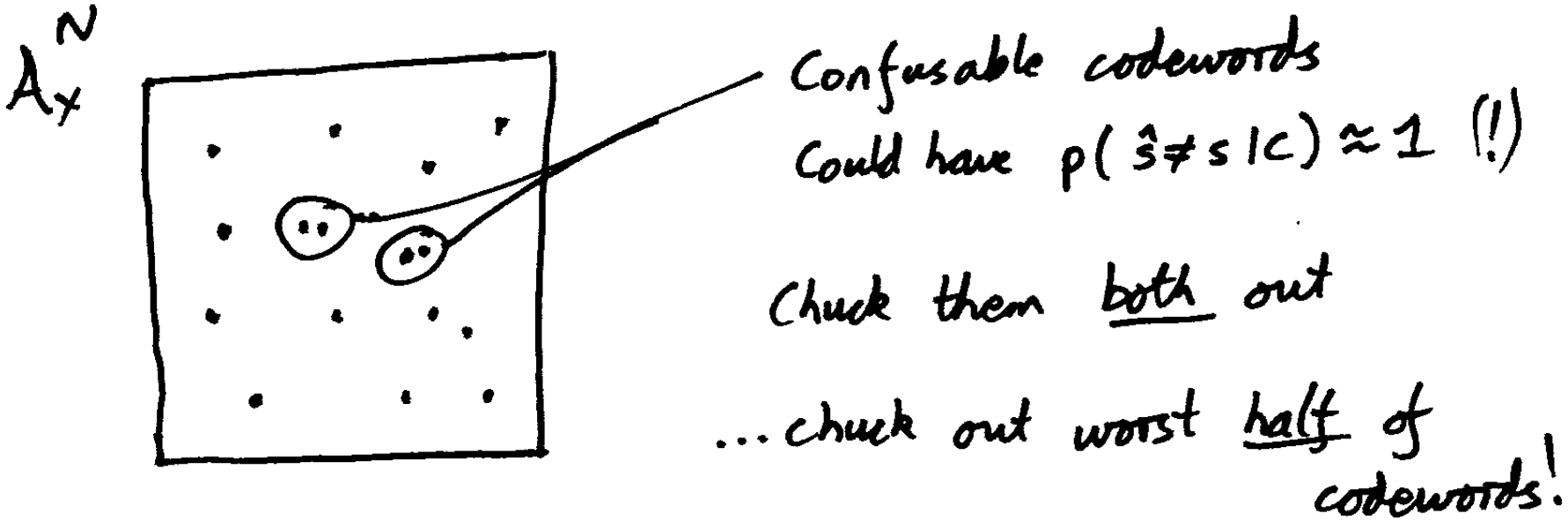
$$\langle p_B \rangle \equiv \sum_{\mathcal{C}} P(\hat{s} \neq s \,|\, \mathcal{C})\, P(\mathcal{C}) < 2\delta$$

Some codes will have error rates more/less than this

There exists *a* code with block error:

$$p_B(\mathcal{C}) \equiv P(\hat{s} \neq s \,|\, \mathcal{C}) < 2\delta$$

Worst case codewords



Maximal block error: $p_{BM}(\mathcal{C}) \equiv \max_s P(\hat{s} \neq s \,|\, s, \mathcal{C})$ could be close to 1.

$p_{BM} < 4\delta$ for expurgated code.

Now have $2^{NR'-1}$ codewords, rate $= R' - 1/N$.

Noisy channel coding theorem

For N large enough, can shrink β's and δ's close to zero.

For large N a code exists with rate close to C with error close to zero. (As close as you like for large enough N.)

In the 'week 7' notes we showed that it is impossible to transmit at rates greater than the capacity, without non-negligible probability of error for particular channels. This is also true in general.

Code distance

Distance, $d \equiv \min_{s,s'} \left|\mathbf{x}^{(s)} - \mathbf{x}^{(s')}\right|$

E.g., $d=3$ for the $[7,4]$ Hamming code

Can *always* correct $\lfloor (d-1)/2 \rfloor$ errors

Distance of random codes?

$\left|\mathbf{x}^{(s)} - \mathbf{x}^{(s')}\right| \approx \frac{N}{2}$ for large N

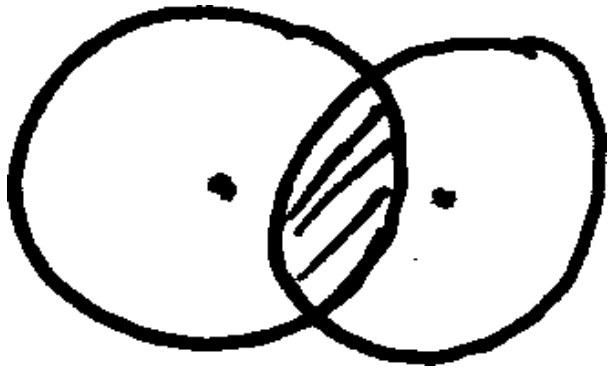
Not *guaranteed* to correct errors in $\geq \frac{N}{4}$ bits

With BSC get $\approx Nf$ errors, and proof works for $f > \frac{1}{4}$

Distance isn't everything

Distance can sometimes be a useful measure of a code

However, good codes have codewords that aren't separated by twice the number of errors we want to correct

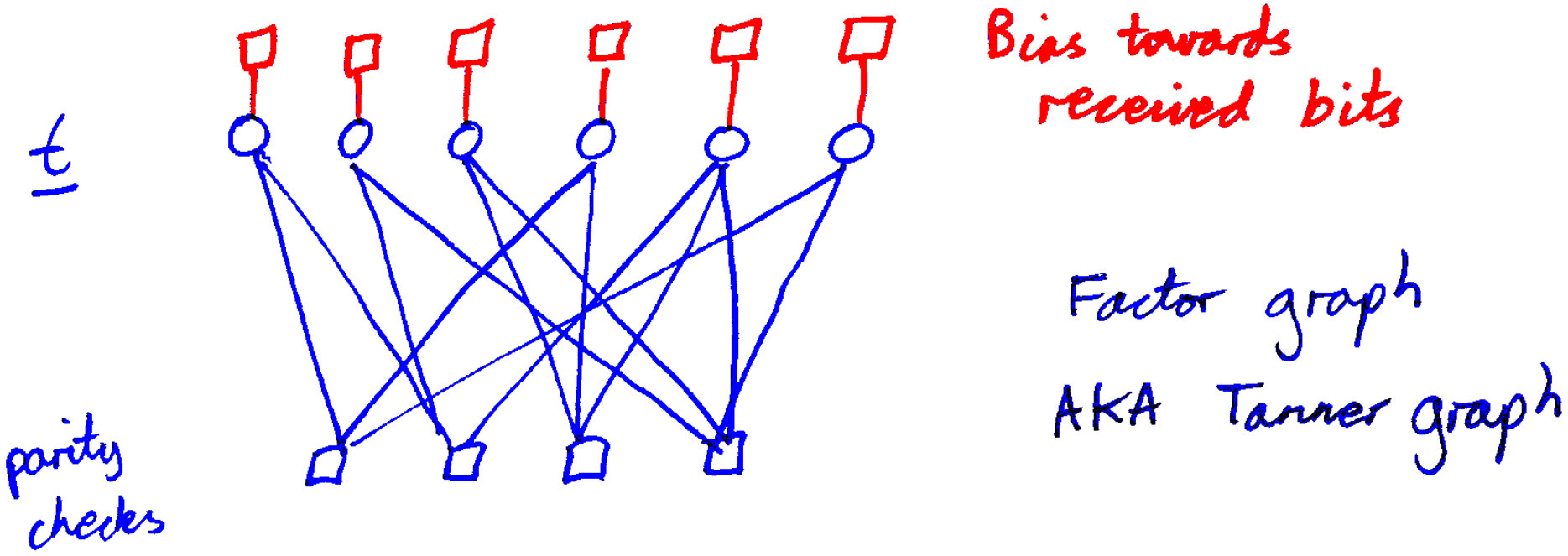


In high-dimensions the overlapping volume is tiny.

Shannon-limit approaching codes for the BSC correct *almost all* patterns with Nf errors, even though they can't strictly correct *all* such patterns.

Low Density Parity Check codes

LDPC codes originally discovered by Gallagher (1961)
Sparse graph codes like LDPC not used until 1990s.



Prior over codewords $P(\mathbf{t}) \propto \mathbb{I}(H\mathbf{t} = \mathbf{0})$
Posterior over codewords $P(\mathbf{t} \,|\, \mathbf{r}) \propto P(\mathbf{t})\, Q(\mathbf{r} \,|\, \mathbf{t})$

Why Low Density Parity Check (LDPC) codes?

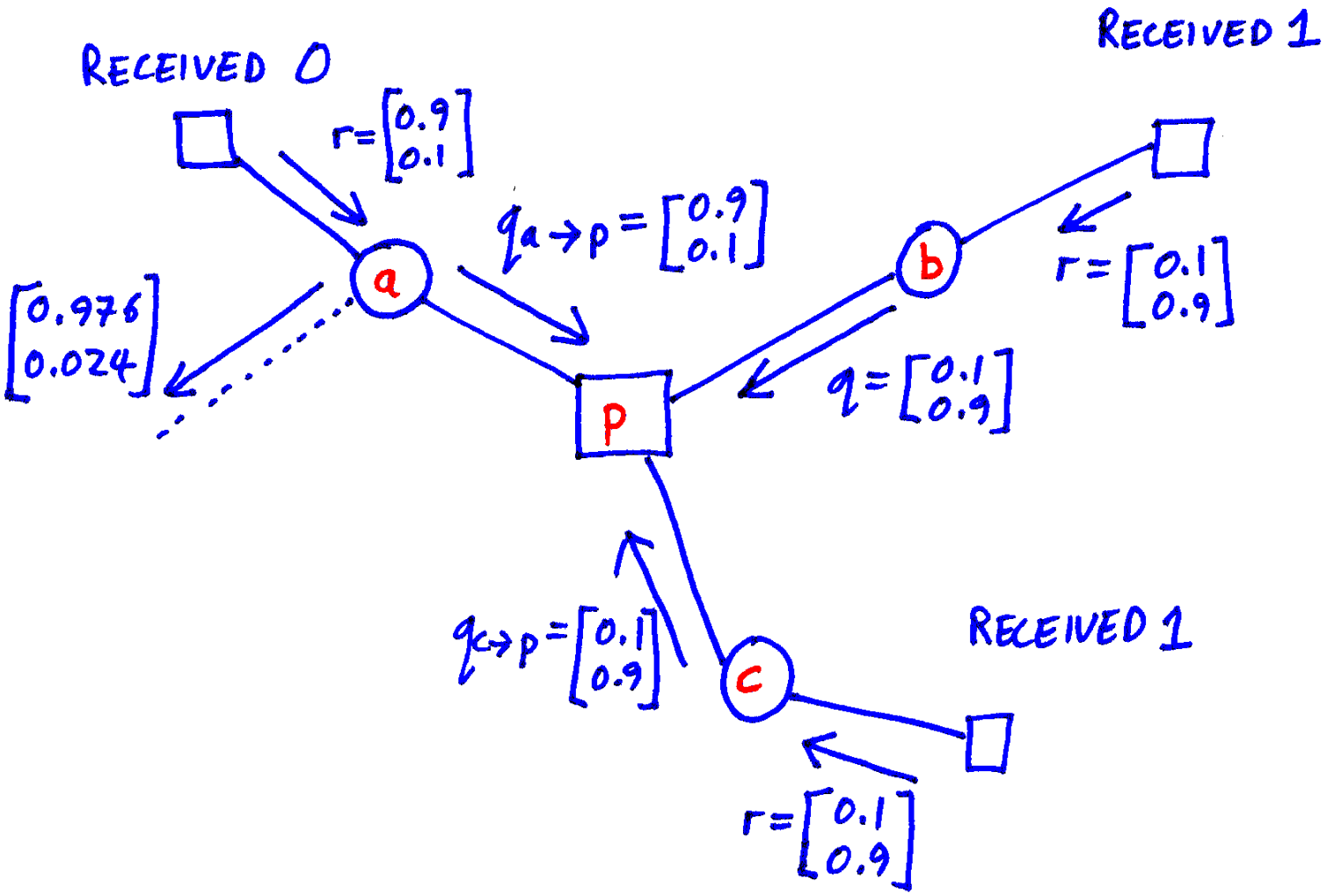
For some channels, the noisy channel coding theorem can be reproved for randomly generated linear codes. However, not all ways of generating *low-density* codes, with each variable only involved in a few parity checks and vice-versa, are very good.

For some sequences of low-density codes, the Shannon limit is approached for large block-lengths.

For both uniformly random linear codes, or random LDPC codes, the results are for optimal decoding: $\hat{\mathbf{t}} = \text{argmax}_{\mathbf{t}} P(\mathbf{t} \,|\, \mathbf{r})$. This is a hard combinatorial optimization problem in general. The reason to use low-density codes is that we have good approximate solvers: use the sum-product algorithm (AKA "loopy belief propagation") — decode if the thresholded beliefs give a setting of $\mathbf{t}$ that satisfies all parity checks.

Sum-Product algorithm

Example with three received bits and one parity check



p336 MacKay, p399 Bishop "Pattern recognition and machine learning"

Sum-Product algorithm notes:

Beliefs are combined by element-wise multiplying
Two types of messages: variable $\rightarrow$ factor and factor $\rightarrow$ variable
Messages combine beliefs from all neighbours except recipient

Variable $\rightarrow$ factor:

$$q_{n\rightarrow m}(x_n) = \prod_{m' \in \mathcal{M}(n)\setminus m} r_{m'\rightarrow n}(x_n)$$

Factor $\rightarrow$ variable:

$$r_{m\rightarrow n}(x_n) = \sum_{\mathbf{x}_m \setminus n} \left(f_m(\mathbf{x}_m) \prod_{n' \in \mathcal{N}(m)\setminus n} q_{n'\rightarrow m}(x_{n'}) \right)$$

Example $r_{p\rightarrow a}$ in diagram, with sum over $(b, c) \in \{(0,0),(0,1),(1,0),(1,1)\}$
$r_{p\rightarrow a}(0) = 1\times0.1\times0.1 + 0 + 0 + 1\times0.9\times0.9 = 0.82$
$r_{p\rightarrow a}(1) = 0 + 1\times0.1\times0.9 + 1\times0.9\times0.1 + 0 = 0.18$

More Sum-Product algorithm notes:

Messages can be renormalized, e.g. to sum to 1, at any time.

I did this for the outgoing message from a to an imaginary factor downstream. This message gives the relative beliefs about about the settings of a given the graph we can see:

$$b_n(x_n) = \prod_{m' \in \mathcal{M}(n)} r_{m' \to n}(x_n)$$

The settings with maximum belief are taken and, if they satisfy the parity checks, used as the decoded codeword.

The beliefs are the correct posterior marginals if the factor graph is a tree. Empirically the decoding algorithm works well on low-density graphs that aren't trees. Loopy belief propagation is also sometimes used in computer vision and machine learning, however, it will not give accurate or useful answers on all inference/optimization problems!

We haven't covered efficient implementation which uses Fourier transform tricks to compute the sum quickly.