

# Teoria dell'informazione - Appunti di Federico Torrielli

## Disclaimer per il lettore

---

Questo esame non è facile.

Quando ho iniziato a studiarlo, non avevo nient'altro che il libro in inglese.

Per questo ho deciso di scrivere degli appunti completi di spiegazioni facili e dimostrazioni scritte bene. Non sempre tutto è così difficile come gli accademici lo scrivono, la maggior parte delle volte semplicemente balzano i passaggi banali delle dimostrazioni, che per noi studenti sono l'essenziale.

Buon esame, ragazzuoli!

*Federico*, 2022.

## Introduzione: Entropia

---

Misura dell'**incertezza media** di una variabile random. Essa rappresenta il numero medio di bit richiesto per descrivere la variabile random.

$$H(X) = - \sum_x p(x) \log_2 p(x)$$

Nota bene, per le regole dei logaritmi:

$$\log_2 x = \frac{\log x}{\log 2}$$

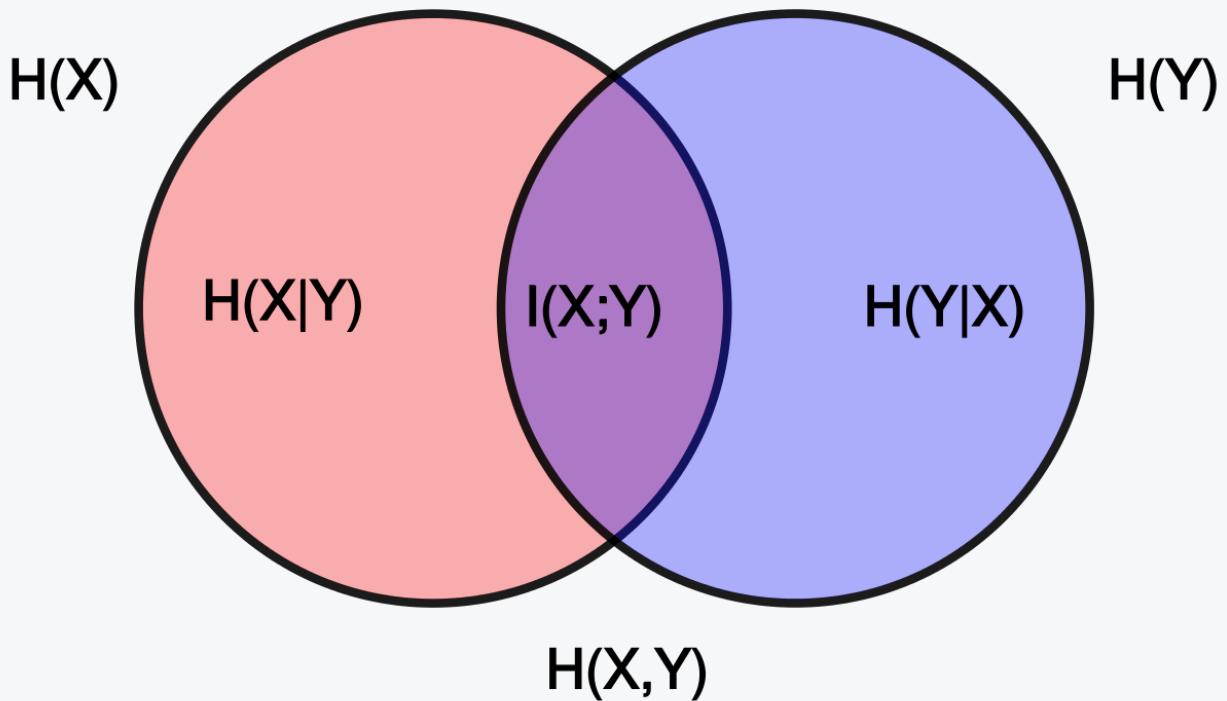
## Esempio entropia

Variabile random con distribuzione uniforme di probabilità su 32 possibili outcome. L'entropia della variabile random, ovvero la lunghezza della stringa che rappresenti tutte le possibili uscite sarà lunga 5 bit, dato che:

$$H(X) = - \sum_{i=1}^{32} p(i) \log p(i) = - \sum_{i=1}^{32} \frac{1}{32} \log \frac{1}{32} = \log 32 = 5$$

## Mutual Information

---



Per due variabili random  $X$  e  $Y$  la riduzione dell'incertezza data da un'altra variabile random è espressa come:

$$I(X;Y) = H(X) - H(X|Y) = \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

Ovvero l'entropia della variabile meno l'entropia condizionale della variabile sull'altra variabile random.

L'informazione mutuale è la **misura della dipendenza** tra due variabili random, essa è:

- Simmetrica
- Non-negativa
- Uguale a 0 se e solo se  $X$  e  $Y$  sono indipendenti

La M.I. è in realtà un caso specifico della più generale quantità chiamata **entropia relativa**, che si scrive come:

$$D(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

Dove  $p, q$  sono probability mass functions.

L'entropia relativa in sé non è una vera e propria metrica, ma possiede alcune proprietà che potrebbero renderla tale come il fatto che sia sempre non-negativa e che sia 0 se e solo se  $p = q$ .

## Canale di comunicazione

Un **CDC** è un sistema nel quale l'output dipende probabilisticamente dal suo input. Esso è caratterizzato da una matrice di transizione di probabilità  $p(y|x)$  che determina la distribuzione condizionale dell'output dato l'input.

La **capacità** di questo canale per l'input  $X$  e l'output  $Y$  è:

$$C = \max_{p(x)} I(X; Y)$$

La capacità è anche il **rateo massimo** entro il quale possiamo mandare l'informazione nel canale e recuperare l'informazione dall'output con una probabilità di errore minima.

## Recap: cosa usiamo dove

---

- **Data compression**: L'entropia  $H$  di una variabile random è il **lower bound** della lunghezza media della più corta descrizione di una variabile random. Possiamo costruire descrizioni con una lunghezza media entro 1 bit dall'entropia.
- **Data transmission**: considereremo il problema di trasmettere l'informazione così che il ricevente possa fare decoding del messaggio con una piccolissima probabilità di errore. Essenzialmente vogliamo ritrovare delle **codewords** che sono mutualmente lontane: le loro versioni *rumorose* (noisy versions) devono essere perciò distinguibili. Shannon ha scoperto che possiamo mandare dell'informazione ad ogni rateo sotto la capacità  $C$  del canale con una arbitrariamente piccola probabilità di errore.

# Information Measures: intro

## Indipendenza e catene di Markov

---

Concetti base della probabilità che devono essere rivisti. Qua parleremo di variabili sempre discrete.

### Indipendenza (2.1)

Due variabili random  $X$  ed  $Y$  sono indipendenti, ovvero  $X \perp Y$ , se

$$\forall x, y \quad p(x, y) = p(x)p(y)$$

### Mutual Independence (2.2)

Per  $n \geq 3$  variabili random, esse sono mutualmente indipendenti se

$$\forall x_1, x_2, \dots, x_n \quad p(x_1, x_2, x_n) = p(x_1)p(x_2)\dots p(x_n)$$

### Pairwise Independence (2.3)

Per  $n \geq 3$  variabili random, esse sono pairwise indipendenti se  $X_i$  e  $X_j$  sono indipendenti  $\forall 1 \leq i < j \leq n$

### Indipendenza condizionale (2.4)

Per delle variabili random  $X, Y, Z$ , Sappiamo che  $X$  è indipendente da  $Z$  condizionando su  $Y$ , denotato come  $X \perp Z|Y$ , se:

$$p(x, y, z)p(y) = p(x, y)p(y, z)$$

### Catena di Markov (2.6)

Per delle variabili random  $X_1, X_2, \dots, X_n$  dove  $n \geq 3$ ,  $X_1 \rightarrow X_2 \rightarrow \dots \rightarrow X_n$  forma una catena di Markov se:

$$p(x_1, x_2, \dots, x_n)p(x_2)p(x_3)\dots p(x_{n-1}) = p(x_1, x_2)p(x_2, x_3)\dots p(x_{n-1}, x_n)$$

### Conseguenza della catena di Markov (2.7)

$X_1 \rightarrow X_2 \rightarrow \dots \rightarrow X_n$  forma una catena di Markov se e solo se  $X_n \rightarrow X_{n-1} \rightarrow \dots \rightarrow X_1$  forma una catena di Markov.

*Proof.* Questo è direttamente collegato alla simmetria inherente alla definizione della catena di Markov in (2.15).

# Information measures: Shannon I.M.

## Entropia

---

$$H(X) = - \sum_x p(x) \log p(x)$$

Dando per scontata la definizione già data precedentemente dell'entropia secondo Shannon, sappiamo di poterla scrivere anche come:

$$H(X) = -E \log p(X)$$

Dove  $E$  sarebbe l'attesa di una certa funzione, che possiamo scrivere per una funzione generica  $g(x)$  come:

$$Eg(x) = \sum_x p(x)g(x)$$

L'entropia  $H(X)$  di una certa variabile random  $X$  è, in altri termini, la quantità media di *incertezza* rimossa una volta rivelato l'outcome di  $X$ .

## Entropia binaria

---

Per  $0 \leq \gamma \leq 1$  definiamo l'entropia binaria come:

$$h_b(\gamma) = -\gamma \log \gamma - (1 - \gamma) \log(1 - \gamma)$$

Sapendo che per  $X$  con distribuzione  $\{\gamma, 1 - \gamma\}$ :

$$H(X) = h_b(\gamma)$$

## Entropia connessa (joint entropy)

---

La  $H(X, Y)$  di una coppia di variabili random è definita come:

$$H(X, Y) = - \sum_{x,y} p(x, y) \log p(x, y) = -E \log p(X, Y)$$

E da questa formula possiamo ricavare:

$$H(Y|X) = \sum_x p(x) \left[ - \sum_y p(y|x) \log p(y|x) \right]$$

Dove la somma interna è l'entropia di  $Y$  condizionata su una  $x$  fissata. Quindi a questo punto possiamo riscrivere l'entropia condizionata come:

$$H(Y|X) = \sum_x p(x) H(Y|X=x)$$

Dove

$$H(Y|X=x) = - \sum_y p(y|x) \log p(y|x)$$

E quindi se vogliamo fare joint + condizionata:

$$H(Y|X, Z) = \sum_z p(z) H(Y|X, Z=z)$$

Dove:

$$H(Y|X, Z=z) = - \sum_{x,y} p(x, y|z) \log p(y|x, z)$$

## Joint Entropy: Venn diagram proof

$$H(X, Y) = H(X) + H(Y|X)$$

e

$$H(X, Y) = H(Y) + H(X|Y)$$

*Proof:*

$$\begin{aligned} H(X, Y) &= -E \log p(X, Y) \\ &= -E \log[p(X)p(Y|X)] \\ &= -E \log p(X) - E \log p(Y|X) \\ &= H(X) + H(Y|X). \end{aligned}$$

## Mutual Information

---

$$I(X; Y) = \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} = E \log \frac{p(X, Y)}{p(X)p(Y)}$$

L'informazione mutuale tra  $X$  e se stessa è l'entropia di  $X$  stessa:  $I(X; X) = H(X)$ .

*Proof:*

$$\begin{aligned} I(X; X) &= E \log \frac{p(X)}{p(X)^2} \\ &= -E \log p(X) \\ &= H(X) \end{aligned}$$

## Proposizione 2.19

$$\begin{aligned} I(X; Y) &= H(X) - H(X|Y) \\ I(X; Y) &= H(Y) - H(Y|X) \end{aligned}$$

e

$$I(X; Y) = H(X) + H(Y) - H(X, Y)$$

## Mutual Information (condizionando su Z)

$$I(X; Y|Z) = \sum_{x,y,z} p(x, y, z) \log \frac{p(x, y|z)}{p(x|z)p(y|z)} = E \log \frac{p(X, Y|Z)}{p(X|Z)p(Y|Z)}.$$

L'informazione mutuale tra  $X$  e se stessa (condizionando su  $Z$ ) è l'entropia di  $X$  condizionando su  $Z$ :  $I(X; X|Z) = H(X|Z)$ .

## Proposizione 2.22

$$\begin{aligned} I(X; Y|Z) &= H(X|Z) - H(X|Y, Z) \\ I(X; Y|Z) &= H(Y|Z) - H(Y|X, Z) \end{aligned}$$

e

$$I(X; Y|Z) = H(X|Z) + H(Y|Z) - H(X, Y|Z)$$

## Chain Rules (2.4)

---

Andiamo qui ad esplorare le chain rules, molto usate in teoria dell'informazione e probabilità. Per lunghezza ho omesso le dimostrazioni.

### Chain Rule per l'entropia

$$H(X_1, X_2, \dots, X_n) = \sum_{i=1}^n H(X_i | X_1, \dots, X_{i-1})$$

### Chain Rule per entropia condizionata

$$H(X_1, X_2, \dots, X_n | Y) = \sum_{i=1}^n H(X_i | X_1, \dots, X_{i-1}, Y)$$

### Chain Rule per la mutual information

$$I(X_1, X_2, \dots, X_n; Y) = \sum_{i=1}^n I(X_i; Y | X_1, \dots, X_{i-1})$$

### Chain Rule per la conditional mutual information

$$I(X_1, X_2, \dots, X_n; Y | Z) = \sum_{i=1}^n I(X_i; Y | X_1, \dots, X_{i-1}, Z)$$

## Informational Divergence (Kullback–Leibler divergence)

---

Per sapere quanto sono diverse due distribuzioni di probabilità  $p$  e  $q$  su un comune alfabeto  $X$  utilizziamo la seguente formula:

$$D(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = E_p \log \frac{p(X)}{q(X)}$$

### Fundamental Inequality (2.29)

$$\ln a \leq a - 1$$

Con l'uguaglianza se  $a = 1$

*Proof:* Sia  $f(a) = \ln a - a + 1$ . Sia poi  $f'(a) = 1/a - 1$  e  $f''(a) = -1/a^2$ .

Visto che  $f(1) = 0$ ,  $f'(1) = 0$  e  $f''(1) = -1 < 0$ , vediamo che  $f(a)$  ha come valore massimo 0 solo quando  $a = 1$ .

### Corollario 2.30

$$\begin{aligned}\forall a > 0 \\ \ln a \geq 1 - \frac{1}{a}\end{aligned}$$

*Proof:* Sostituendo semplicemente  $a$  con  $1/a$ , la dimostrazione diventa esattamente uguale alla precedente.

## Divergence Inequality

Per ogni due distribuzioni di probabilità  $p$  e  $q$  su un alfabeto comune  $X$ :

$$D(p||q) \geq 0$$

Con l'uguaglianza se e solo se  $p = q$ .

*Proof:* se  $q(x) = 0$  per qualche  $x \in S_p$ , allora  $D(p||q) = \infty$  e il teorema è banalmente vero.

Ordunque, assumiamo che  $q(x) > 0$  per tutti gli  $x \in S_p$ , allora:

$$\begin{aligned}D(p||q) &= (\log e) \sum_{x \in S_p} p(x) \ln \frac{p(x)}{q(x)} \\ &\geq (\log e) \sum_{x \in S_p} p(x) \left(1 - \frac{q(x)}{p(x)}\right) \\ &= (\log e) \left[ \sum_{x \in S_p} p(x) - \sum_{x \in S_p} q(x) \right] \\ &\geq 0.\end{aligned}$$

## Diseguaglianze di base (non-negatività)

In questo sotto-capitolo proveremo che tutte le information measures di Shannon sono sempre **non-negative**, ovvero che queste quantità sono non-negative per tutte le distribuzioni di probabilità congiunte per le variabili random coinvolte.

### Non-negatività della mutual information

$$I(X; Y|Z) \geq 0,$$

Con l'uguaglianza se e solo se  $X$  e  $Y$  sono indipendenti condizionando su  $Z$

*Proof:* si osserva che

$$\begin{aligned}I(X; Y|Z) &= \sum_{x,y,z} p(x, y, z) \log \frac{p(x, y|z)}{p(x|z)p(y|z)} \\ &= \sum_z p(z) \sum_{x,y} p(x, y|z) \log \frac{p(x, y|z)}{p(x|z)p(y|z)} \\ &= \sum_z p(z) D(p_{XY|z} || p_{X|z} p_{Y|z})\end{aligned}$$

Dove ho usato  $p_{XY|z}$  per denotare  $\{p(x, y|z), (x, y) \in X \times Y\}$ , etc.

E visto che abbiamo uno  $z$  fissato, scopriamo che:

$$D(p_{XY|z} || p_{X|z} p_{Y|z}) \geq 0.$$

## Non-negatività dell'entropia ed entropia condizionale

Considera l'entropia di  $H(X)$  di una variabile random:  $\forall x \in S_x$ , visto che  $0 < p(x) \leq 1, \log p(x) \leq 0$ , è chiaro dunque che  $H(X) \geq 0$ .

### Proposizione 2.35

$$H(X) = 0 \text{ sse } X \text{ deterministica}$$

*Proof:*

- Se  $X$  deterministica,  $\exists x^* \in X \rightarrow p(x^*) = 1 \text{ and } p(x) = 0 \forall x \neq x^*$ , allora  $H(X) = -p(x^*) \log p(x^*) = 0$ .
  - Se invece  $X$  non deterministica,  $\exists x^* \in X \rightarrow 0 < p(x^*) < 1$  allora  $H(X) \geq -p(x^*) \log p(x^*) > 0$ .
- Dunque concludiamo che  $H(X) = 0$  sse  $X$  deterministica.

### Proposizione 2.36

$$H(Y|X) = 0 \text{ sse } Y \text{ funzione di } X$$

*Proof:* Abbiamo già visto che  $H(Y|X) = 0$  sse  $H(Y|X = x) = 0$ , e questo accade solo quando  $Y$  p deterministica per ogni  $x$ , ovvero  $Y$  è funzione di  $X$

### Proposizione 2.37

$$I(X; Y) = 0 \text{ sse } X, Y \text{ indipendenti}$$

## Piccoli teoremi sulle disuguaglianze di informazioni

---

### Condizionare non aumenta l'entropia

$$H(Y|X) \leq H(Y)$$

Con l'uguaglianza sse  $X$  e  $Y$  sono indipendenti.

*Proof:* possiamo provare lo statement considerando che

$$H(Y|X) = H(Y) - I(X; Y) \leq H(Y)$$

Che è vera dato che  $I$  è sempre *non-negativa*, essa è *stretta* sse  $I(X; Y) = 0$ .

### Limite dell'indipendenza per l'entropia

$$H(X_1, X_2, \dots, X_n) \leq \sum_{i=1}^n H(X_i)$$

con l'uguaglianza sse  $X_i, \forall i$  sono mutualmente indipendenti.

*Proof:* data la chain rule per l'entropia,

$$\begin{aligned} H(X_1, X_2, \dots, X_n) &= \sum_{i=1}^n H(X_i | X_1, \dots, X_{i-1}) \\ &\leq \sum_{i=1}^n H(X_i) \end{aligned}$$

E questo segue dalla diseguaglianza dato che condizionare non aumenta l'entropia.

### △ aumenta l'informazione mutuale

$$I(X; Y, Z) \geq I(X; Y)$$

### Lemma 2.41

Se  $X \rightarrow Y \rightarrow Z$  formano una catena di Markov, allora:

$$\begin{aligned} I(X; Z) &\leq I(X; Y) \\ I(X; Z) &\leq I(Y; Z) \end{aligned}$$

La cui dimostrazione resta sul libro.

### Data processing theorem

Se  $U \rightarrow X \rightarrow Y \rightarrow V$  formano una catena di Markov, allora:

$$I(U; V) \leq I(X; Y)$$

*Proof:* assumendo vero  $U \rightarrow X \rightarrow Y \rightarrow V$ , allora, per le proprietà delle catene di Markov, viste prima, abbiamo  $U \rightarrow X \rightarrow Y$  e  $U \rightarrow Y \rightarrow V$ .

Dalla prima catena di Markov, per il teorema precedente otteniamo  $I(U; Y) \leq I(X; Y)$ , mentre dalla seconda otteniamo  $I(U; V) \leq I(U; Y)$ , che se combinate fanno ottenere il teorema.

## Diseguaglianza di Fano

---

### Upper bound di una variabile sul suo alfabeto (che utilizzeremo per Fano dopo), Teorema 2.43

Per ogni variabile random  $X$  su un alfabeto  $\chi$ :

$$H(X) \leq \log |\chi|$$

Dove la diseguaglianza è stretta sse  $X$  è distribuita uniformemente su  $\chi$ .

*Proof:* sia  $u$  una distribuzione uniforme su  $\chi$ , quindi  $u(x) = |\chi|^{-1} \forall x$ .

$$\begin{aligned}
& \log |\chi| - H(X) \\
&= - \sum_{x \in S_x} p(x) \log |\chi|^{-1} + \sum_{x \in S_x} p(x) \log p(x) \\
&= - \sum_{x \in S_x} p(x) \log u(x) + \sum_{x \in S_x} p(x) \log p(x) \\
&= \sum_{x \in S_x} p(x) \log \frac{p(x)}{u(x)} \\
&= D(p||u) \\
&\geq 0.
\end{aligned}$$

## Corollario 2.44

L'entropia di una variabile random può essere qualsiasi valore non-negativo reale.  
La *proof* si trova al punto 2.44, pagina 32.

## Disuguaglianza di Fano

Siano  $X$  e  $\hat{X}$  variabili random con lo stesso alfabeto  $\chi$ , allora:

$$H(X|\hat{X}) \leq h_b(P_e) + P_e \log(|\chi| - 1)$$

Dove  $h_b$  è l'entropia binaria.

*Proof:* definiamo una variabile random come segue:

$$Y = 0 \text{ if } X = \hat{X}, 1 \text{ if } X \neq \hat{X}$$

E una variabile  $Y$  come indicatore dell'evento di errore  $X \neq \hat{X}$  con  $P(Y = 1) = P_e$  e  $H(Y) = h_b(P_e)$ . Visto che  $Y$  è una funzione di  $X$  e  $\hat{X}$ :

$$H(Y|X, \hat{X}) = 0$$

Allora:

$$\begin{aligned}
& H(X|\hat{X}) \\
&= H(X|\hat{X}) + H(Y|X, \hat{X}) \\
&= H(X, Y|\hat{X}) \\
&= H(Y|\hat{X}) + H(X|\hat{X}, Y) \\
&\leq H(Y) + H(X|\hat{X}, Y) \\
&= H(Y) + \sum_{\hat{x} \in \chi} [Pr(\hat{X} = \hat{x}, Y = 0) \cdot H(X|\hat{X} = \hat{x}, Y = 0) + Pr(\hat{X} = \hat{x}, Y = 1) \cdot H(X|\hat{X} = \hat{x}, Y = 1)] 
\end{aligned}$$

E dato che  $H(X) = 0$  sse  $X$  deterministica (2.35), allora:

$$H(X|\hat{X} = \hat{x}, Y = 0) = 0$$

E se  $\hat{X} = \hat{x}$  e  $Y = 1$  allora  $X$  deve prendere un valore nel set  $\{x \in \chi : x \neq \hat{x}\}$  che contiene  $|\chi| - 1$  elementi, quindi per il teorema 2.43 sull'upper bound di una variabile sul suo alfabeto otteniamo:

$$H(X|\hat{X} = \hat{x}, Y = 1) \leq \log(|\chi| - 1),$$

Dove il suo upper bound non dipende da  $\hat{x}$ , dunque:

$$\begin{aligned}
 H(X|\hat{X}) &\leq h_b(P_e) + \left( \sum_{\hat{x} \in \chi} \Pr(\hat{X} = \hat{x}, Y = 1) \right) \log(|\chi| - 1) \\
 &= h_b(P_e) + \Pr(Y = 1) \log(|\chi| - 1) \\
 &= h_b(P_e) + P_e \log(|\chi| - 1)
 \end{aligned}$$

Che ne completa la dimostrazione.  $\square$

## Disegualanza di Jensen

---

Se  $f$  è una funzione convessa e  $X$  è una variabile random,

$$Ef(X) \geq f(EX)$$

Inoltre, se  $f$  è *strettamente convessa*, l'uguaglianza implica che  $X = EX$  con probabilità 1 (ad esempio, se  $X$  è una costante)

*Proof:* Proviamo questo teorema per distribuzioni discrete per *induzione* sul numero dei punti di massa.

Per una distribuzione a due punti di massa la disegualanza diventa:

$$p_1 f(x_1) + p_2 f(x_2) \geq f(p_1 x_1 + p_2 x_2),$$

Che segue direttamente dalla definizione di funzioni convesse.

Supponiamo che il teorema sia *vero* per distribuzioni con  $k - 1$  punti di massa. Allora scrivere  $p'_i = p_i / (1 - p_k)$  per  $i = 1, 2, \dots, k - 1$ , abbiamo

$$\begin{aligned}
 \sum_{i=1}^k p_i f(x_i) &= p_k f(x_k) + (1 - p_k) \sum_{i=1}^{k-1} p'_i f(x_i) \\
 &\geq p_k f(x_k) + (1 - p_k) f\left(\sum_{i=1}^{k-1} p'_i x_i\right) \\
 &\geq f\left(p_k x_k + (1 - p_k) \sum_{i=1}^{k-1} p'_i x_i\right) \\
 &= f\left(\sum_{i=1}^k p_i x_i\right),
 \end{aligned}$$

Dove la prima disegualanza segue direttamente dall'ipotesi induttiva e la seconda segue dalla definizione di convessità.  $\square$

## Funzione convessa

Una funzione  $f(x)$  è *convessa* su un intervallo  $(a, b)$  se per ogni  $x_1, x_2 \in (a, b)$  e  $0 \leq \lambda \leq 1$  abbiamo,

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2)$$

## Funzione concava

Una funzione  $f$  è concava se  $-f$  è convessa.

## Proof alternative derivate dalla Diseguaglianza di Jensen

---

**Theorem 1.6**  $I(X; Y)$  is a convex function of the input probabilities  $p(x)$ .

*Proof* We think of the transition probabilities  $p(y|x)$  as being fixed, and consider two input random variables  $X_1$  and  $X_2$  with probability distributions  $p_1(x)$  and  $p_2(x)$ . If  $X$ 's probability distribution is a convex combination  $p(x) = \alpha p_1(x) + \beta p_2(x)$ , we must show that

$$\alpha I(X_1; Y_1) + \beta I(X_2; Y_2) \leq I(X; Y),$$

where  $Y_1$ ,  $Y_2$  and  $Y$  are the channel outputs corresponding to  $X_1$ ,  $X_2$ , and  $X$ , respectively. To do this consider the following manipulation, which uses obvious notational shorthand:

$$\begin{aligned} & \alpha I(X_1; Y_1) + \beta I(X_2; Y_2) - I(X; Y) \\ &= \sum_{x,y} \alpha p_1(x, y) \log \frac{p(y|x)}{p_1(y)} + \sum_{x,y} \beta p_2(x, y) \log \frac{p(y|x)}{p_2(y)} \quad (\text{see Eq.(1.7)}) \\ &\quad - \sum_{x,y} [\alpha p_1(x, y) + \beta p_2(x, y)] \log \frac{p(y|x)}{p(y)} \\ &= \alpha \sum_{x,y} p_1(x, y) \log \frac{p(y)}{p_1(y)} + \beta \sum_{x,y} p_2(x, y) \log \frac{p(y)}{p_2(y)}. \end{aligned} \tag{1.12}$$

We now apply Jensen's inequality to each of the above sums. For example,

$$\sum_{x,y} p_1(x, y) \log \frac{p(y)}{p_1(y)} \leq \log \sum_{x,y} p_1(x, y) \frac{p(y)}{p_1(y)}.$$

But

$$\begin{aligned} \sum_{x,y} p_1(x, y) \frac{p(y)}{p_1(y)} &= \sum_y \frac{p(y)}{p_1(y)} \sum_x p_1(x, y) \\ &= \sum_y \frac{p(y)}{p_1(y)} \cdot p_1(y) \\ &= 1. \end{aligned}$$

Hence the first sum in (1.12) is  $\leq 0$ ; similarly, so is the second.  $\square$

**Corollary** The entropy function  $H(p_1, p_2, \dots, p_r)$  is convex.

*Proof* Let  $X$  be a random variable distributed according to  $P\{X = i\} = p_i$ . Then  $I(X; X) = H(X) = H(p_1, p_2, \dots, p_r)$ . The result now follows from Theorem 1.6.  $\square$

**Theorem 1.7**  $I(X; Y)$  is convex  $\cup$  in the transition probabilities  $p(y|x)$ .

*Proof* Here the input probabilities  $p(x)$  are fixed, but we are given two sets of transition probabilities  $p_1(y|x)$  and  $p_2(y|x)$  and a convex combination  $p(y|x) = \alpha p_1(y|x) + \beta p_2(y|x)$ . It is required to show that

$$I(X; Y) \leq \alpha I(X; Y_1) + \beta I(X; Y_2), \quad (1.13)$$

where  $Y, Y_1, Y_2$  are the channel outputs corresponding to the transition probabilities  $p(y|x)$ ,  $p_1(y|x)$ , and  $p_2(y|x)$ . Again using obvious notation, the difference between the left and right sides of (1.13) is (see Eq. (1.5))

$$\begin{aligned} & \sum_{x,y} [\alpha p_1(x, y) + \beta p_2(x, y)] \log \frac{p(x|y)}{p(x)} \\ & - \sum_{x,y} \alpha p_1(x, y) \log \frac{p_1(x|y)}{p(x)} - \sum_{x,y} \beta p_2(x, y) \log \frac{p_2(x|y)}{p(x)} \\ & = \alpha \sum_{x,y} p_1(x, y) \log \frac{p(x|y)}{p_1(x|y)} + \beta \sum_{x,y} p_2(x, y) \log \frac{p(x|y)}{p_2(x|y)}. \end{aligned} \quad (1.14)$$

The first sum in (1.14) is, by Jensen's inequality,

$$\begin{aligned} & \leq \alpha \log \left[ \sum_{x,y} p_1(x, y) \frac{p(x|y)}{p_1(x|y)} \right] \\ & = \alpha \log \left[ \sum_{x,y} p(x|y) p_1(y) \right] \\ & = \alpha \log \sum_y p_1(y) = 0. \end{aligned}$$

Similarly the second sum is  $\leq 0$ .  $\square$

## Esercizio 1

Si disponga di due monete, una normale e una con entrambe le facce ‘testa’.

Si scelga casualmente una delle monete e si lanci consecutivamente per 2 volte.

Considerando le variabili aleatorie  $M$  (moneta scelta) e  $C$  (numero di facce ‘testa’ uscite nei 2 lanci), calcolare la mutua informazione  $I(M, C)$ .

Ricordiamo inizialmente la formula della mutua informazione:

$$I(M, C) = \sum_{m,c} p(m, c) \log_2 \frac{p(m, c)}{p(m)p(c)}$$

In questo caso  $p(m, c)$  è da fare su tutte le possibili combinazioni delle variabili aleatorie  $M$  e  $C$ . Vediamole insieme:

- $M$  può essere un evento che ha  $p(M = m_1) = p(M = m_2) = 1/2$ , dato che non è stato specificato che la moneta è truccata, immaginiamo che sia *fair*, e quindi che cada da una parte o dall'altra con egual probabilità
- $C$  può essere un evento che può avere 0,1 oppure 2 teste (moltiplichiamo tra di loro eventi indipendenti):
  - $P(C = 2) = P(\text{testa}) \cdot P(\text{testa}) = 0.75^2 = 0.5625$
  - $P(C = 0) = \dots = 0.25$
  - $P(C = 1) = P(\text{testa}) \cdot P(\text{croce}) = 0.75 \cdot 0.25 = 0.1875$

Vediamo tutte le possibili combinazioni:

$$p(M = m_1, C = 0) \log \frac{p(M = m_1, C = 0)}{p(M = m_1)p(C = 0)} = 0.25 \log \frac{0.25}{0.5 \cdot 0.25} = 0.25$$

$$p(M = m_1, C = 1) \log \frac{p(M = m_1, C = 1)}{p(M = m_1)p(C = 1)} = 0.25 \log \frac{0.25}{0.5 \cdot 0.1875} \approx 0.35375$$

$$p(M = m_1, C = 2) \log \frac{p(M = m_1, C = 2)}{p(M = m_1)p(C = 2)} = 0.25 \log \frac{0.25}{0.5 \cdot 0.5625} \approx -0.04248$$

$$p(M = m_2, C = 2) \log \frac{p(M = m_2, C = 2)}{p(M = m_2)p(C = 2)} = 1$$

Ho qui escluso i casi con  $0 \log 0$  che restituiscono 0 e sono ininfluenti sulla somma.

La somma ci da approssimativamente 1.56.

Se volessimo calcolare anche l'entropia, allora, sapendo che la formula è

$$H(X) = - \sum_x p(x) \log_2 p(x)$$

Possiamo applicarla alle singole v.a. nel modo seguente:

$$\begin{aligned} H(M) &= -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1 \text{ bit} \\ H(C) &= 1.42 \text{ bits} \end{aligned}$$

## Esercizio 2

Un test di ammissione all'università verte su una prova di fisica o una di matematica.

Si stima che un candidato abbia una probabilità fallire la prova di fisica con probabilità 0,5 e di superare la prova di matematica con probabilità 0,8.

Si supponga di estrarre a sorte con eguale probabilità una delle due prove.

Se il candidato viene ammesso all'università, quanta informazione fornisce questo evento sul fatto che sia stata rispettivamente estratta la prova di fisica o quella di matematica?

Definiamo le variabili aleatorie utili allo svolgimento dell'esercizio:

- $S$  = *successo*
- $E$  = *viene estratta la prova i-esima*

$$\begin{aligned} P(S|E = \text{fisica}) &= 0.5 \\ P(S|E = \text{mate}) &= 0.8 \\ P(E = \text{fisica}, \text{mate}) &= 0.5 \\ P(S) &= 0.5 \cdot 0.5 + 0.5 \cdot 0.8 = 0.65 \end{aligned}$$

L'esercizio vuole sapere quanto sia probabile che avendo passato il test, abbia influito l'una o l'altra materia!

$$P(E = \text{mate}|S) = \frac{P(S|E = \text{mate})P(E = \text{mate})}{P(S)} = 0.6153\dots$$

e

$$P(E = \text{fisica}|S) = \dots = 0.3846\dots$$

## Esercizio 3

*Coin flips.* A fair coin is flipped until the first head occurs. Let  $X$  denote the number of flips required.

- (a) Find the entropy  $H(X)$  in bits. The following expressions may be useful:

$$\sum_{n=0}^{\infty} r^n = \frac{1}{1-r}, \quad \sum_{n=0}^{\infty} n r^n = \frac{r}{(1-r)^2}.$$

Se  $X$  è il numero di lanci prima che esca testa e la moneta è regolare, allora  $X \approx Geometrica(0.5)$ .

Sapendo che  $P(X) = 0.5^k$  allora:

$$H(X) = E(-\log P(X))$$

ovvero

$$H(X) = \sum_{k=1}^{\infty} -\log(0.5^k)0.5^k$$

E per le regole dei logaritmi ( $\log_a(x^b) = b \log_a(x)$  quindi  $-\log_2(0.5^k) = k$ ):

$$H(X) = \sum_{k=1}^{\infty} k0.5^k$$

E quindi, per la regola data qui sopra:

$$\frac{0.5}{(1-0.5)^2} = 2$$

che rimane uguale all'attesa  $1/p$  della geometrica, dato che  $1/0.5 = 2$ .

# Zero-error data compression

## The entropy bound

In questa sezione l'obiettivo è stabilire che  $H(X)$  è il lower bound fondamentale della **lunghezza attesa** del numero di simboli che servono per descrivere l'evento  $X$ : questo è detto l'**entropy bound**.

### Source Code

Un source code **D-ario**  $C$  per una variabile sorgente random  $X$  è un *mapping* da  $\chi$  a  $D^*$ , il set di tutte le sequenze finite di simboli per un alfabeto **D-ario**.

Se si considera una sorgente d'informazione  $\{X_k, k \geq 1\}$  dove  $X_k$  sono variabili random che prendono tutti i valori dallo stesso alfabeto, applichiamo il source code  $C$  ad ogni  $X_k$  e concateniamo le *codewords*. Una volta che abbiamo concatenato le cw, i limiti delle cw non sono più esplicativi, ovvero, quando  $C$  viene applicata alla sequenza sorgente, una sequenza di output viene prodotta, dove le codewords non sono più riconoscibili.

Noi, però, siamo interessati negli *uniquely decodable codes*, che definiamo ora.

## Source Coding Theorem (recap)

Il teorema stabilisce che, per una serie di variabili aleatorie indipendenti ed identicamente distribuite (i.i.d.) di lunghezza che tende ad infinito, non è possibile comprimere i dati in un messaggio più corto dell'entropia totale senza perdita di informazione. Al contrario, compressioni arbitrariamente vicine al valore di entropia sono possibili, con probabilità di perdita di informazione piccola a piacere.

### Lunghezza attesa di un codice

$$L = \sum_i p_i l_i$$

Ovvero la sommatoria di tutte le probabilità per le loro lunghezze dei codici.

## Uniquely decodable codes

Un codice  $C$  è uniquely decodable se per ogni sequenza finita sorgente, la sequenza di simboli di codice che corrisponde alla sorgente è differente da qualsiasi altra sequenza che potrebbe corrispondere a qualsiasi altra sequenza finita.

Se il codice  $C$  è u.d. vuol dire che dato l'output possiamo sempre recuperare l'input! Un codice, per essere u.d., deve rispettare la *Kraft Inequality*.

### ESEMPIO NON-UNIQUELY DECODABLE

Sia  $\chi = \{A, B, C, D\}$ . Sia il codice  $C$  definito come:

$x$	$C(x)$
$A$	0
$B$	1
$C$	01
$D$	10.

Le sequenze  $AAD, ACA, AABA$  producono tutte 0010, quale genera quale, dato il codice?

### Kraft Inequality

Sia  $C$  un source code **D-ario**, siano  $l_1, l_2, \dots, l_m$  le lunghezze delle codewords. Se  $C$  è u.d., allora vale la seguente diseguaglianza:

$$\sum_{k=1}^m D^{-l_k} \leq 1.$$

*Proof:* sia  $N$  un numero intero positivo arbitrario, si consideri che:

$$\left(\sum_{k=1}^m D^{-l_k}\right)^N = \sum_{k_1=1}^m \sum_{k_2=1}^m \dots \sum_{k_N=1}^m D^{-(l_{k_1}+l_{k_2}+\dots+l_{k_N})}$$

Raccogliendo la parte destra dell'equazione, otteniamo:

$$\left(\sum_{k=1}^m D^{-l_k}\right)^N = \sum_{i=1}^{Nl_{max}} A_i D^{-i}$$

Dove:

$$l_{max} = \max_{1 \leq k \leq m} l_k$$

E  $A_i$  restituisce il numero totale di sequenze di  $N$  codewords con lunghezza totale di  $i$  simboli. Visto che il codice è **u.d.**, queste sequenze devono essere per forza distinte, e quindi vale

$$A_i \leq D^i$$

Visto che ci sono  $D^i$  distinte sequenze di  $i$  code symbols.

Se andiamo a sostituire questo alla prima equazione (quella dove abbiamo raccolto la parte destra):

$$\left(\sum_{k=1}^m D^{-l_k}\right)^N \leq \sum_{i=1}^{Nl_{max}} 1 = Nl_{max}$$

E questo vale  $\forall N$  che tende ad infinito, dunque  $\square$ .

## Entropy Bound (Shannon's source coding theorem)

Sia  $C$  un u.d. code D-ario da una variabile  $X$  di entropia  $H_D(X)$ , allora per la lunghezza attesa di  $C$ ,  $L$ , vale :

$$L \geq H_D(X)$$

Il lower bound è stretto sse  $l_i = -\log_D p_i \forall i$ .

*Questo teorema ci dice, in parole povere, che la lunghezza di un codice u.d. è al minimo l'entropia della sorgente.*

*Proof:* dato che  $C$  è u.d., le lunghezze delle sue codewords soddisfano la diseguaglianza di Kraft, ovvero

$$L = \sum_i p_i \log_D D^{l_i}$$

E richiamando la definizione di entropia, ovvero

$$H_D(X) = - \sum_i p_i \log_D p_i$$

Otteniamo dunque:

$$\begin{aligned} L - H_D(X) &= \sum_i p_i \log_D (p_i D^{l_i}) \\ &= (\ln D)^{-1} \sum_i p_i \ln(p_i D^{l_i}) \\ &\geq (\ln D)^{-1} \sum_i p_i \left(1 - \frac{1}{p_i D^{l_i}}\right) \\ &= (\ln D)^{-1} \left[ \sum_i p_i - \sum_i D^{-l_i} \right] \\ &\geq (\ln D)^{-1} (1 - 1) \\ &= 0 \end{aligned}$$

Alla 3a riga abbiamo utilizzato la fundamental inequality, alla penultima la Kraft inequality. In modo che il lower bound sia stretto, queste due inequality devono esserlo contemporaneamente. La fundamental è stretta sse  $p_i D^{l_i} = 1$  oppure  $l_i = -\log_D p_i \forall i$ . Se questo è vero, otteniamo:

$$\sum_i D^{-l_i} = \sum_i p_i = 1$$

E dunque anche la Kraft è stretta, perciò  $\square$ .

### Corollario 4.7

$$H(X) \leq \log |\chi|$$

*Proof:* immaginiamo di dover fare encoding di ogni outcome di una variabile random  $X$  con un simbolo distinto nel gruppo  $\{1, 2, \dots, \chi\}$ . Ovviamente stiamo parlando di un codice  $|\chi|$ -ario u.d. con una lunghezza attesa di 1. Come entropy bound, avremo:

$$H_{|\chi|}(X) \leq 1$$

ovvero

$$H(X) \leq \log |\chi|$$

## Ridondanza di un codice

La **ridondanza**  $R$  di un u.d. code **D-ario** è la differenza tra la lunghezza attesa del codice e l'entropia della sorgente. Essa è sempre non-negativa nel caso di u.d. codes.

## Prefix Codes

---

Un codice è detto **prefix-free** quando non esiste una codeword che sia un prefisso di ogni altra codeword. Per brevità, chiameremo i codici **prefix-free prefix codes**.

### Teorema 4.11

Esiste un prefix code **D-ario** con lunghezza delle codewords  $l_1, l_2, \dots, l_m$  se la Kraft inequality è soddisfatta:

$$\sum_{k=1}^m D^{-l_k} \leq 1$$

*Proof:* dobbiamo provare l'esistenza di un prefix code D-ario con lunghezze  $l_1, l_2, \dots, l_m$  se queste lunghezze soddisfano la diseguaglianza di Kraft.

Consideriamo dunque tutte le sequenze D-arie di lunghezza minore o uguale a  $l_m$  e guardiamole in ottica di nodi di un albero pieno D-ario di profondità  $l_m$ .

Ci riferiremo alla sequenza di lunghezza  $l$  come al nodo di ordine  $l$ .

La nostra strategia è di scegliere nodi come codeword in ordine crescente sulle lunghezze delle codewords. Nello specifico, scegliamo un nodo di ordine  $l_1$  come prima codeword, poi un nodo di ordine  $l_2$  come seconda, e così via, per permettere che ogni codeword nuova non abbiamo come prefisso le codeword precedenti. Se riusciamo a scegliere tutte le  $m$  codewords, allora il set risultante di codewords forma un prefix code con il set di lunghezze desiderato.

Ci sono dunque  $D^{l_1} > 1$  nodi di ordine  $l_1$  che possono essere scelti come prima codeword. Quindi scegliere la prima codeword è **sempre** possibile.

Assumiamo quindi che le prime  $i$  codeword siano state scelte con successo, e che quindi vogliamo scegliere un nodo di ordine  $l_{i+1}$  come la  $i + 1$ -esima codeword tale che non abbia come prefisso nessuna delle codeword precedentemente scelte. In altre parole, il  $i + 1$ -esimo nodo da essere scelto **non** deve essere un discendente di qualsiasi altra codeword scelta.

Osserviamo che per  $1 \leq j \leq i$ , la codeword con lunghezza  $l_j$  ha  $D^{l_{i+1}-l_j}$  discendenti di ordine  $l_{i+1}$ . Visto che tutte le codeword precedentemente scelte non sono prefisse, i discendenti di ordine  $l_{i+1}$  non si sovrappongono. Dunque, una volta notato il fatto che il numero totale di nodi di ordine  $l_{i+1}$  è  $D^{l_{i+1}}$ , il numero di nodi che possono essere scelti come la  $i + 1$ -esima codeword è

$$D^{l_{i+1}} - D^{l_{i+1}-l_1} - \dots - D^{l_{i+1}-l_i}$$

E se  $l_1, l_2, \dots, l_m$  soddisfano la diseguaglianza di Kraft, abbiamo

$$D^{-l_1} + \dots + D^{-l_i} + D^{-l_{i+1}} \leq 1.$$

Moltiplicando per  $D^{l_{i+1}}$  e ri-organizzando i termini, abbiamo che

$$D^{l_{i+1}} - D^{l_{i+1}-l_1} - \dots - D^{l_{i+1}-l_i} \geq 1.$$

La parte sinistra della disequazione ci conta il numero di nodi che possono essere scelti per la  $i+1$ -esima codeword. Quindi ciò è possibile, come mostrato.  $\square$

## Corollario 4.12

Esiste un prefix code **D-ario** che raggiunge l'entropy bound per una distribuzione  $\{p_i\}$  se e solo se questa è **D-adica**.

*Proof:* Consideriamo un codice prefisso D-ario che arriva all'entropy bound per una distribuzione  $\{p_i\}$ . Sia  $l_i$  la lunghezza delle codewords assegnate alla probabilità  $p_i$ . Per il teorema sull'entropy bound (ovvero,  $L \geq H(x)$ ) per tutti gli  $i$ , abbiamo che  $l_i = -\log_D p_i$  scritto anche  $p_i = D^{-l_i}$ . Quindi la nostra distribuzione  $\{p_i\}$  è D-adica.

La dimostrazione inversa resta sul libro (1), a pagina 88.

## Huffman Codes

---

Un Huffman code è un **codice ottimale**, ovvero che possiede la lunghezza attesa  $E$  minima, costruito attraverso la **procedura di Huffman**. In generale esiste più di un codice ottimale e qualche codice ottimale **non** può essere costruito con tale procedura.

Per semplicità, trattiamo prima di codici di Huffman **binari**: un binary prefix code per una source  $X$  con distribuzione  $\{p_i\}$  è rappresentato da un binary code tree, dove ogni foglia dell'albero ne corrisponde ad una codeword. La procedura di Huffman consiste semplicemente nel formare un codetree in modo che la lunghezza attesa sia minima.

### Huffman procedure

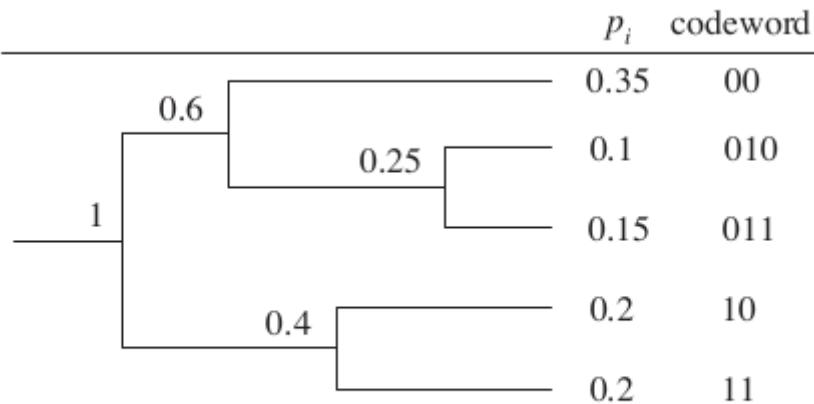
Continuiamo ad unire le due più piccole masse di probabilità fino a quando non ne abbiamo solo più una.

Il merging di due masse di probabilità corrisponde alla formazione di un nodo interno nel code tree. Vediamo un esempio:

### Huffman procedure example

Sia  $X$  la sorgente con  $\chi = \{A, B, C, D, E\}$  e con probabilità rispettivamente di 0.35, 0.1, 0.15, 0.2, 0.2. Nel primo step, andiamo a fare merge delle probability masses 0.1 e 0.15 in una p.m. 0.25. Nel secondo step, mergiamo le due p.m. 0.2 in una 0.4. Nel terzo

step, mergiamo la 0.35 e 0.25 in una 0.6 e finalmente nell'ultimo step facciamo la 0.6 e 0.4 in una p.m. 1. Il code tree sarà il seguente:



**Fig. 4.2.** The Huffman procedure.

Se la grandezza dell'alfabeto è  $m$ , allora la procedura di Huffman ci mette  $m - 1$  step per completarsi nella costruzione di un codice *binario*.

**Spiegazione dumb style:** partiamo dalle foglie, uniamo i numerini più piccoli tra di loro, questo sarà il nodo padre, come si vede nell'immagine ( $0.1+0.15$  fa il nodo padre  $0.25$ ). Così via fino a quando non arriviamo alla radice. Quando abbiamo costruito l'albero, non ci resta che tornare indietro e mettere uno 0 quando il path è sinistra, e 1 quando il path va a destra (o il contrario, va bene tutto dai), dalla radice alla foglia. Ad esempio, 0.35 sarà destra destra quindi scriveremo 0 mentre l'ultimo 0.2 sarà sinistra sinistra quindi scriveremo 11. Facile, no?

### Huffman: codeword più corte a probabilità più grandi

*In un codice ottimale, le codeword più corte sono abbinate a probabilità più grandi.*

**Proof:** consideriamo  $1 \leq i \leq j \leq m$  tale che  $p_i > p_j$ . Assumiamo che in un codice le codewords  $c_i$  e  $c_j$  siano tali che  $l_i > l_j$ , ovvero, la codeword più corta viene assegnata alla probabilità più piccola. Scambiando dunque  $c_i$  con  $c_j$ , la lunghezza attesa del codice sarà:

$$(p_i l_j + p_j l_i) - (p_i l_i + p_j l_j) = (p_i - p_j)(l_j - l_i) < 0$$

E visto che  $p_i > p_j$  e  $l_i > l_j$ , il codice è ancora migliorabile e quindi non ottimale,  $\square$ .

### Lemma 4.16

Esiste un codice ottimale nel quale le codewords assegnate alle più piccole probabilità sono *siblings*, ovvero, le codeword hanno la stessa lunghezza e differiscono solo nell'ultimo simbolo.

**Proof:** consideriamo un codice ottimale a caso. Dall'ultimo lemma abbiamo imparato che la codeword  $c_m$  assegnata alla probabilità  $p_m$  ha la lunghezza massima. In questo

caso il fratello di  $c_m$  non può essere il prefisso di alcun'altra codeword.

Però, nel dire questo, affermiamo anche che il fratello di  $c_m$  sia una codeword.

Assumiamo dunque che non lo sia. Allora possiamo rimpiazzare  $c_m$  col suo *parent* per migliorare il codice dato che la lunghezza della codeword assegnata da  $p_m$  è ridotta di 1, mentre tutte le altre codewords rimangono così come sono. Questa, però, è una contraddizione all'assunzione che il codice sia ottimale, quindi il *fratello* di  $c_m$  deve per forza essere una codeword.

Se il fratello di  $c_m$  viene assegnato a  $p_{m-1}$  allora il codice ha già dei fratelli. Altrimenti, assumiamo che il fratello di  $c_m$  sia assegnato a  $p_i$ , dove  $i < m - 1$ . Visto che  $p_i \geq p_{m-1}$  sappiamo che  $l_{m-1} \geq l_i = l_m$ . Dall'altro punto di vista, invece, per il lemma dell'ottimalità degli huffman code,  $l_{m-1}$  è sempre minore o uguale a  $l_m$ , che implica che  $l_{m-1} = l_m = l_i$ . Dunque possiamo scambiare le codewords per  $p_i$  e  $p_{m-1}$  senza cambiare la lunghezza attesa del codice, per ottenere il codice ottimale,  $\square$ .

Mostriamo qui di seguito un breve esempio dal libro:

Suppose  $c_i$  and  $c_j$  are siblings in a code tree. Then  $l_i = l_j$ . If we replace  $c_i$  and  $c_j$  by a common codeword at their parent, call it  $c_{ij}$ , then we obtain a reduced code tree, and the probability of  $c_{ij}$  is  $p_i + p_j$ . Accordingly, the probability set becomes a reduced probability set with  $p_i$  and  $p_j$  replaced by a probability  $p_i + p_j$ . Let  $L$  and  $L'$  be the expected lengths of the original code and the reduced code, respectively. Then

$$L - L' = (p_i l_i + p_j l_j) - (p_i + p_j)(l_i - 1) \quad (4.34)$$

$$= (p_i l_i + p_j l_i) - (p_i + p_j)(l_i - 1) \quad (4.35)$$

$$= p_i + p_j, \quad (4.36)$$

which implies

$$L = L' + (p_i + p_j). \quad (4.37)$$

This relation says that the difference between the expected length of the original code and the expected length of the reduced code depends only on the values of the two probabilities merged but not on the structure of the reduced code tree.

## Ottimalità della procedura di Huffman

La procedura di Huffman produce sempre un prefix code **ottimale**.

*Proof:* consideriamo un codice ottimale nel quale  $c_m$  e  $c_{m-1}$  sono fratelli. Questo codice ottimale esiste per il lemma appena visto. Sia  $\{p'_i\}$  il probability set ridotto ottenuto da  $\{p_i\}$  unendo  $p_m$  con  $p_{m-1}$ . Dalla formuletta 4.37 dell'immagine abbiamo imparato che  $L'$  è la lunghezza attesa del codice ottimale per  $\{p_i\}$ . Quindi, se riusciamo a trovare il codice ottimale per  $\{p'_i\}$  possiamo utilizzarlo per costruire il codice ottimale di  $\{p_i\}$ . Facendo il merge di  $p_m$  e  $p_{m-1}$  la grandezza del problema si riduce di uno. Ripetiamo le

riduzioni fino a quando non otteniamo 2 codewords di lunghezza 1, dunque la procedura di Huffman produce veramente dei codici ottimali,  $\square$ .

### Teorema 4.18

La lunghezza attesa di un Huffman code, scritta come  $L_{huff}$  soddisfa la seguente disequazione:

$$L_{huff} < H_D(X) + 1$$

Questo limite è il più stretto tra tutti i limiti superiori su  $L_{huff}$  che dipendono solamente dall'entropia della sorgente.

*Proof:* costruiamo ora un prefix code con una lunghezza attesa minore di  $H(X) + 1$ . A quel punto, perché un Huffman code è un prefix code ottimale,  $L_{huff}$  avrà un upper bound di  $H(X) + 1$ .

Consideriamo la costruzione del prefix code con delle lunghezze di codewords  $\{l_i\}$ , dove:

$$l_i = \lceil -\log_D p_i \rceil.$$

Quindi (4.40):

$$-\log_D p_i \leq l_i < -\log_d p_i + 1$$

Che possiamo scrivere anche come:

$$p_i \geq D^{-l_i} > D^{-1} p_i$$

E quindi:

$$\sum_i D^{-l_i} \leq \sum_i p_i = 1,$$

Ovvero  $\{l_i\}$  soddisfa la diseguaglianza di Kraft, che implica che è possibile costruire un prefix code con lunghezze  $\{l_i\}$ .

Rimane però da mostrare che  $L < H(X) + 1$ . Quindi, consideriamo:

$$\begin{aligned} L &= \sum_i p_i l_i \\ &< \sum_i p_i (-\log_D p_i + 1) \\ &= -\sum_i p_i \log_D p_i + \sum_i p_i \\ &= H(X) + 1, \end{aligned}$$

Dove la seconda formula mostrata segue dall'upper bound mostrato in 4.40. Dunque concludiamo che:

$$L_{huff} \leq L < H(X) + 1.$$

Ma per mostrare che questo limite è il più stretto esistente dobbiamo mostrare che esiste una sequenza di distribuzioni  $P_k$  tale che  $L_{\text{Huff}}$  si avvicina a  $H(X) + 1$  per  $k \rightarrow \infty$ . Questo è possibile farlo considerando:

$$P_k = \left\{ 1 - \frac{D-1}{k}, \frac{1}{k}, \dots, \frac{1}{k} \right\}, \text{ where } k \geq D$$

L'Huffman code per ogni  $P_k$  consiste in  $D$  codewords di lunghezza 1. Quindi  $L_{\text{Huff}}$  è uguale ad 1 per ogni  $k$ . Mentre  $k$  si approssima a infinito,  $H(X) \rightarrow 0$ , dunque  $L_{\text{Huff}}$  si avvicina a  $H(X) + 1$ ,  $\square$ .

Il codice che qui abbiamo costruito per provare il teorema è detto uno **Shannon Code**: l'idea è quella che per far sì che il codice sia quanto più vicino ad essere ottimale, dobbiamo scegliere  $l_i$  vicino a  $-\log p_i \forall i$ . Quando  $\{p_i\}$  è D-adica,  $l_i$  può essere scelta come esattamente  $-\log p_i$ , perché questi ultimi sono interi: in questo caso l'entropy bound è stretto.

From the entropy bound and the above theorem, we have

$$H(X) \leq L_{\text{Huff}} < H(X) + 1. \quad (4.49)$$

Now suppose we use a Huffman code to encode  $X_1, X_2, \dots, X_n$  which are  $n$  i.i.d. copies of  $X$ . Let us denote the length of this Huffman code by  $L_{\text{Huff}}^n$ . Then (4.49) becomes

$$nH(X) \leq L_{\text{Huff}}^n < nH(X) + 1. \quad (4.50)$$

Dividing by  $n$ , we obtain

$$H(X) \leq \frac{1}{n} L_{\text{Huff}}^n < H(X) + \frac{1}{n}. \quad (4.51)$$

As  $n \rightarrow \infty$ , the upper bound approaches the lower bound. Therefore,  $n^{-1}L_{\text{Huff}}^n$ , the coding rate of the code, namely the average number of code symbols needed to encode a source symbol, approaches  $H(X)$  as  $n \rightarrow \infty$ . But of course, as  $n$  becomes large, constructing a Huffman code becomes very complicated. Nevertheless, this result indicates that entropy is a fundamental measure of information.

## Esercizi

### Esercizio 1: quante dita ha un marziano?

2. How many fingers has a Martian? Let

$$S = \begin{pmatrix} S_1, \dots, S_m \\ p_1, \dots, p_m \end{pmatrix}.$$

The  $S_i$ 's are encoded into strings from a  $D$ -symbol output alphabet in a uniquely decodable manner. If  $m = 6$  and the codeword lengths are  $(l_1, l_2, \dots, l_6) = (1, 1, 2, 3, 2, 3)$ , find a good lower bound on  $D$ . You may wish to explain the title of the problem.

**Solution:** *How many fingers has a Martian?*

97

In questo problema basta guardare per quale  $D$  vale la *Kraft Inequality* con le length date.

In questo caso, dopo aver provato due, che da 1.75, confermiamo che i marziani hanno 3 dita, visto che:  $3^{-1} + 3^{-1} + 3^{-2} + 3^{-3} + 3^{-2} + 3^{-3} = 0.96$ .

## Esercizio 2

4. Huffman coding. Consider the random variable

$$X = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ 0.49 & 0.26 & 0.12 & 0.04 & 0.04 & 0.03 & 0.02 \end{pmatrix}$$

- (a) Find a binary Huffman code for  $X$ .
- (b) Find the expected codelength for this encoding.
- (c) Find a ternary Huffman code for  $X$ .

**Solution:** *Examples of Huffman codes.*

- (a):

- $x_1 = 0$
- $x_2 = 10$
- $x_3 = 110$
- $x_4 = 11100$
- $x_5 = 11101$
- $x_6 = 11110$
- $x_7 = 11111$

- (b):

$$L = \sum p_i l_i = 0.24 \cdot 1 + 0.26 \cdot 2 + 0.12 \cdot 3 + (0.04 + 0.04 + 0.03 + 0.02) \cdot 5 = 177/100 = 1.77$$

.

- (c): boring!

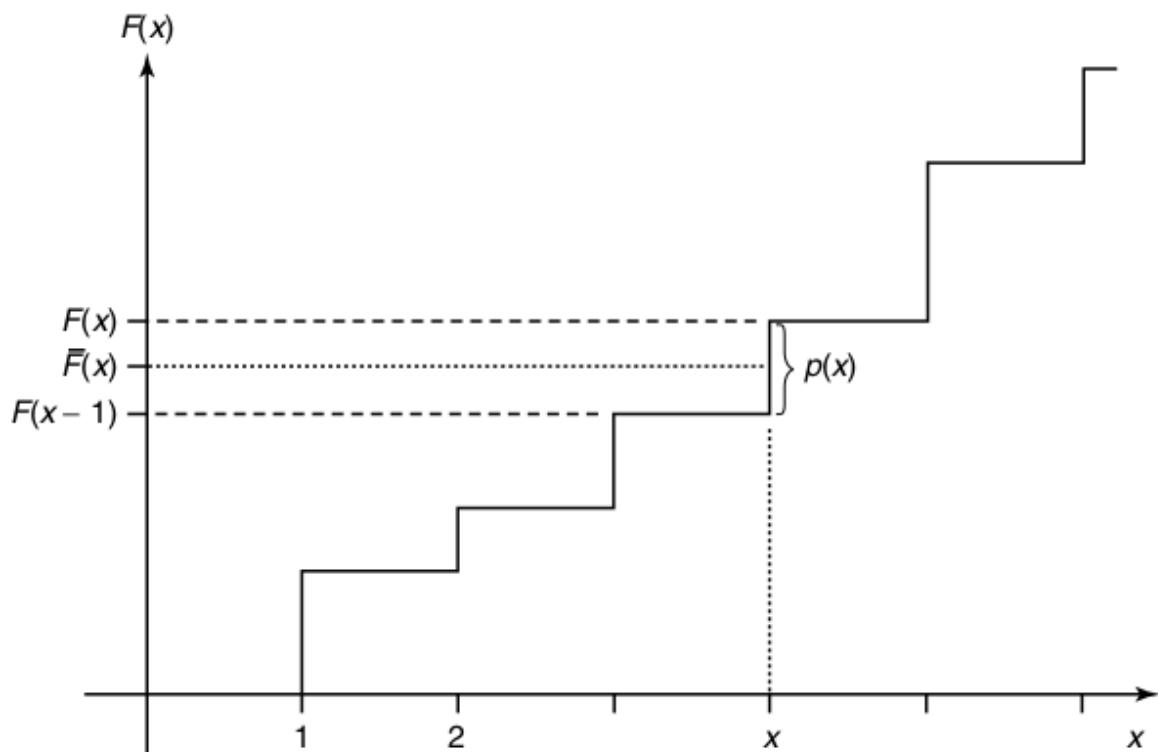
## Shannon-Fano-Elias Coding

In questa sezione descriviamo una procedura costruttiva che usa la funzione di distribuzione cumulativa per assegnare le codewords.

Dato un dizionario  $\chi = \{1, 2, \dots, m\}$ , assumiamo che  $p(x) > 0 \forall x$ . La funzione di distribuzione cumulativa  $F(x)$  è definita come segue:

$$F(X) = \sum_{a \leq x} p(a)$$

Che è esattamente la seguente:



**FIGURE 5.5.** Cumulative distribution function and Shannon–Fano–Elias coding.

Consideriamo ora la funzione di distribuzione cumulativa modificata come segue:

$$\bar{F}(x) = \sum_{a < x} p(a) + \frac{1}{2}p(x)$$

Dove  $\bar{F}(x)$  denota la somma di probabilità di tutti i simboli che sono minori di  $x$  più la metà della probabilità del simbolo  $x$  stesso.

Visto che stiamo parlando di variabili discrete, questa è una funzione a scalino con step size grandi  $p(x)$ . Il valore della funzione  $\bar{F}(x)$  è la metà dello step che corrisponde a  $x$ .

Visto che tutte le probabilità sono positive,  $\bar{F}(a) \neq \bar{F}(b)$  if  $a \neq b$ , e quindi possiamo

determinare  $x$  se conosciamo  $\bar{F}(x)$ . Dunque possiamo usare il valore di questa come codice per  $x$ .

Però, in generale,  $\bar{F}(x)$  è un numero reale che possiamo esprimere solo con un numero infinito di bit, quindi non è proprio efficiente usare il valore esatto di  $\bar{F}(x)$  come codice per  $x$ . Useremo dunque un valore approssimato, ma quale sarà la sua accuratezza?

Assumiamo di troncare  $\bar{F}(x)$  a  $l(x)$  bit (che da ora in poi scriveremo come  $\lfloor \bar{F}(x) \rfloor_{l(x)}$ ). Usiamo quindi i primi  $l(x)$  bit di  $\bar{F}(x)$  come codice per  $x$ , dunque per definizione avremo:

$$\bar{F}(x) - \lfloor \bar{F}(x) \rfloor_{l(x)} < \frac{l}{2^{l(x)}}$$

Se  $l(x) = \lceil \log \frac{1}{p(x)} \rceil + 1$ , allora:

$$\frac{l}{2^{l(x)}} < \frac{p(x)}{2} = \bar{F}(x) - F(x-1)$$

E quindi scopriamo che  $\lfloor \bar{F}(x) \rfloor_{l(x)}$  è tra gli step che corrispondono ad  $x$ , quindi  $l(x)$  bit bastano per descrivere  $x$ .

Dobbiamo a questo punto controllare che il codice sia *prefix-free*. Per farlo dobbiamo guardare che ogni codeword  $z_1, z_2, \dots, z_l$  non rappresenti un punto ma un intervallo  $[0.z_1z_2\dots z_l, 0.z_1z_2\dots z_l + \frac{1}{2^l}]$ : il codice è prefix-free sse gli intervalli che corrispondono alle codewords sono disgiunti.

Ogni intervallo è grande  $2^{-l(x)}$ , che è meno della metà dell'altezza dello step che corrisponde ad  $x$  nella formula  $\frac{l}{2^{l(x)}} < \frac{p(x)}{2} = \bar{F}(x) - F(x-1)$ . La parte finale dell'intervallo è nella bassa metà dello step e l'intervallo che corrisponde alla codeword giace completamente nello step che corrisponde al simbolo della funzione di distribuzione cumulativa. Quindi, gli intervalli sono disgiunti ed il codice è prefix-free.

Visto che utilizziamo  $l(x) = \lceil \log \frac{1}{p(x)} \rceil + 1$  bit per rappresentare  $x$ , la lunghezza attesa  $L$  del codice sarà:

$$L = \sum_x p(x)l(x) = \sum_x p(x)(\lceil \log \frac{1}{p(x)} \rceil + 1) < H(X) + 2.$$

Quindi scopriamo che il coding scheme raggiunge una avg codeword length che sta entro 2 bit dall'entropia!

## SEF Coding for dummies

Data una variabile  $X$  di valori ordinati e  $p(x)$  la probabilità per ogni  $x \in X$ . Definiamo la funzione vista prima:

$$\hat{F}(x) = \sum_{x_i < x} (p(x_i)) + \frac{1}{2}p(x)$$

Dove  $p(x_i)$  sono la probabilità dello step precedente sommate!

Ed eseguiamo l'algoritmo seguente:

```
for x in X:  
    Z = converti in binario F^(x)  
    L(x) = scegli la lunghezza dell'encoding di x attraverso la formula  
(1)  
    code(x) = scegli l'encoding di x, ovvero il primi L(x) bit più  
significativi dopo il punto decimale di Z
```

**Formula (1):**  $\lceil \log_2 \frac{1}{p(x)} \rceil + 1$ .

### ESEMPIO DI ESECUZIONE DETTAGLIATA

Sia  $X = \{A, B, C, D\}$  con probabilità  $p = \{1/3, 1/4, 1/6, 1/4\}$

1. Per A:

1.  $\hat{F}(A) = 0 + \frac{1}{2}p(A) = \frac{1}{2} \cdot \frac{1}{3} = 0.1666\dots$
2.  $Z(A) = 0.001100110011\dots$
3.  $L(A) = \lceil \log_2 \frac{1}{1/3} \rceil + 1 = 3$
4.  $code(A) = \text{primi 3 bit di } Z(A) \text{ scelti da dopo il punto decimale, ovvero 001}$

2. Per B:

1.  $\hat{F}(B) = p(A) + \frac{1}{2}p(B) = \frac{1}{3} + \frac{1}{2} \cdot \frac{1}{4} = \frac{11}{24}$
2.  $Z(B) = 0.01110101\dots$
3.  $L(B) = \lceil \log_2 \frac{1}{1/4} \rceil + 1 = 3$
4.  $code(B) = 011$

3. Per C:

1.  $\hat{F}(C) = p(A) + p(B) + \frac{1}{2} \cdot p(C) = 0.66666\dots$
2.  $Z(C) = 0.1010101\dots$
3.  $L(C) = \lceil \log_2 \frac{1}{1/6} \rceil + 1 = 4$
4.  $code(C) = 1010$

4. Per D:

1.  $\hat{F}(D) = p(A) + p(B) + p(C) + \frac{1}{2}p(D) = 0.875$
2.  $Z(D) = 0.111$
3.  $L(D) = \lceil \log_2 \frac{1}{1/4} \rceil + 1 = 3$
4.  $code(D) = 111$

### Esempio

$x$	$p(x)$	$F(x)$	$\bar{F}(x)$	$\bar{F}(x)$ in Binary	$l(x) = \left\lceil \log \frac{1}{p(x)} \right\rceil + 1$	Codeword
1	0.25	0.25	0.125	0.001	3	001
2	0.5	0.75	0.5	0.10	2	10
3	0.125	0.875	0.8125	0.1101	4	1101
4	0.125	1.0	0.9375	0.1111	4	1111

In questo caso la lunghezza media delle codeword è 2.75 bits e l'entropia è 1.75 bits. L'Huffman code per questo caso resta nell'entropy bound.

### Esempio di costruzione di un SEF code

**Example 5.9.2** We now give another example for construction of the Shannon–Fano–Elias code. In this case, since the distribution is not dyadic, the representation of  $F(x)$  in binary may have an infinite number of bits. We denote  $0.01010101\dots$  by  $0.\overline{01}$ . We construct the code in the following table:

$x$	$p(x)$	$F(x)$	$\bar{F}(x)$	$\bar{F}(x)$ in Binary	$l(x) = \left\lceil \log \frac{1}{p(x)} \right\rceil + 1$	Codeword
1	0.25	0.25	0.125	0.001	3	001
2	0.25	0.5	0.375	0.011	3	011
3	0.2	0.7	0.6	0.1\overline{0011}	4	1001
4	0.15	0.85	0.775	0.1100\overline{011}	4	1100
5	0.15	1.0	0.925	0.1110\overline{110}	4	1110

### Ottimalità competitiva del SEF coding

Abbiamo visto che l'Huffman coding è ottimale e che ha  $L$  minima. Ma cosa possiamo invece dire della performance del SEF per una particolare sequenza?

Consideriamo quindi un gioco a somma-zero con due persone: a due uomini viene data una distribuzione di probabilità e chiesto di creare un codice istantaneo per le distribuzioni. Poi un simbolo è estratto dalla sorgente e il payoff del player può essere solo 1 oppure -1, a seconda che il suo codice sia più corto di quello estratto dall'altra persona. Il payoff è 0 per le patte.

Avere a che fare con le lunghezze di codici di Huffman non è semplice, visto che non esiste un'espressione esplicita delle lunghezze delle codewords, per i codici SEF, invece, sappiamo che le lunghezze delle codewords saranno pari a  $l(x) = \lceil \log \frac{1}{p(x)} \rceil$ .

Dunque possiamo stipulare il seguente teorema:

### Teorema 5.10.1

Siano  $l(x)$  le lunghezze delle codewords associate al codice di Shannon, e siano  $l'(x)$  le lunghezze per ogni altro u.d. code. Allora,

$$Pr(l(X) \geq l'(X) + c) \leq \frac{1}{2^{c-1}}.$$

Per esempio, la probabilità che  $l'(X)$  sia 5 o meno bits più piccolo di  $l(X)$  è  $< \frac{1}{16}$ .

*Proof:*

$$\begin{aligned} Pr(l(X) \geq l'(X) + c) &= Pr(\lceil \log \frac{1}{p(X)} \rceil \geq l'(X) + c) \\ &\leq Pr(\log \frac{1}{p(X)} \geq l'(X) + c - 1) \\ &= Pr(p(X) \leq 2^{-l'(X)-c+1}) \\ &= \sum_{x:p(x) \leq 2^{-l'(X)-c+1}} p(x) \\ &\leq \sum_{x:p(x) \leq 2^{-l'(X)-c+1}} 2^{-l'(x)-(c-1)} \\ &\leq \sum_x 2^{-l'(x)} 2^{-(c-1)} \\ &\leq 2^{-(c-1)} \end{aligned}$$

Visto che  $\sum 2^{-l'(x)} \leq 1$  dalla Kraft inequality.  $\square$

Cosa si capisce da questo?

Che **la maggior parte delle volte nessun codice sarà migliore di quello di Shannon!**

Proviamo ora a rafforzare il risultato ottenuto. In un setting di gioco, ci vorremmo assicurare che  $l(x) < l'(x)$  capiti più spesso di  $l(x) > l'(x)$ . Il fatto che  $l(x) \leq l'(x) + 1$  avvenga con probabilità  $\geq \frac{1}{2}$  non ce lo assicura certamente.

Mostriamo ora che se aggiungiamo un criterio ancor più stretto, il SEF coding è ottimale.

Per farlo, ricordiamo prima che una probability mass function  $p(x)$  è **[diadica]** se  $\log \frac{1}{p(x)}$  è un intero per ogni  $x$ , oppure se la sua **pmf**  $f(n) = 2^{-n}$ ,  $n \in N$ .

## Teorema 5.10.2

Per una  $p(x)$  p.m.f. **diadica**, siano  $l(x) = \log \frac{1}{p(x)}$  le lunghezze di un Shannon code binario per la sorgente, e siano  $l'(x)$  le lunghezze di qualunque altro u.d. code binario per la sorgente, allora:

$$Pr(l(X) < l'(X)) \geq Pr(l(X) > l'(X)),$$

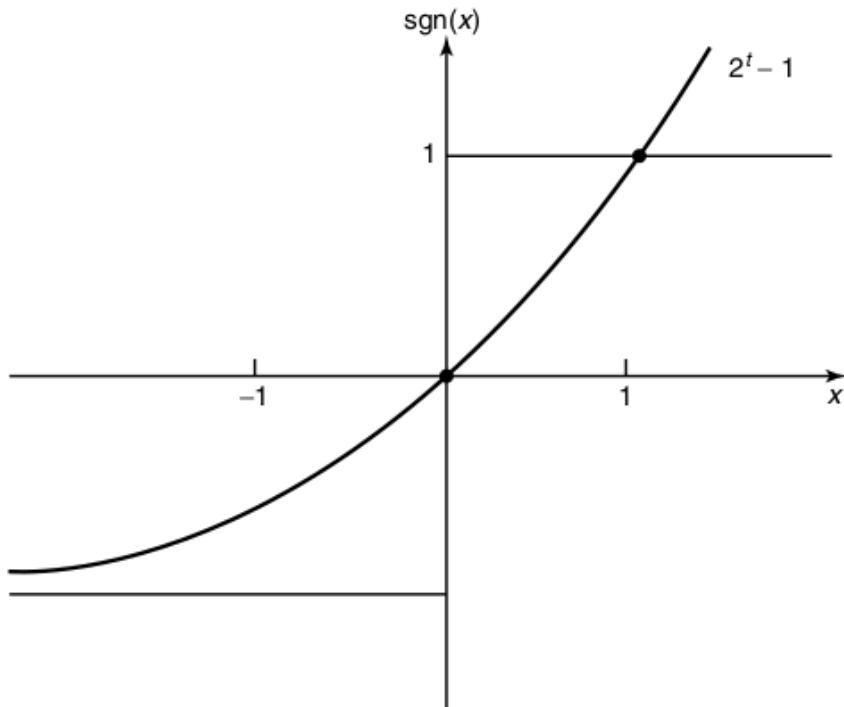
Con l'uguaglianza sse  $l'(x) = l(x) \forall x$ . Dunque, l'assegnamento delle lunghezze  $l(x) = \log \frac{1}{p(x)}$  è competitivamente ottimale.

*Proof:* definiamo una funzione  $sgn(t)$  come segue:

$$\operatorname{sgn}(t) = \begin{cases} 1 & \text{if } t > 0 \\ 0 & \text{if } t = 0 \\ -1 & \text{if } t < 0 \end{cases} \quad (5.86)$$

Then it is easy to see from Figure 5.6 that

$$\operatorname{sgn}(t) \leq 2^t - 1 \quad \text{for } t = 0, \pm 1, \pm 2, \dots \quad (5.87)$$



**FIGURE 5.6.** Sgn function and a bound.

Da notare il fatto che questa diseguaglianza non sia soddisfatta per tutti i  $t$ , ma solo per tutti i valori interi di  $t$ . Possiamo allora scrivere che:

$$\Pr(l'(X) < l(X)) - \Pr(l'(X) > l(X)) = \sum_{x:l'(x) < l(x)} p(x) - \sum_{x:l'(x) > l(x)} p(x) \quad (5.88)$$

$$= \sum_x p(x) \operatorname{sgn}(l(x) - l'(x)) \quad (5.89)$$

$$= E \operatorname{sgn}(l(X) - l'(X)) \quad (5.90)$$

$$\stackrel{(a)}{\leq} \sum_x p(x) (2^{l(x)-l'(x)} - 1) \quad (5.91)$$

$$= \sum_x 2^{-l(x)} (2^{l(x)-l'(x)} - 1) \quad (5.92)$$

$$= \sum_x 2^{-l'(x)} - \sum_x 2^{-l(x)} \quad (5.93)$$

$$= \sum_x 2^{-l'(x)} - 1 \quad (5.94)$$

$$\stackrel{(b)}{\leq} 1 - 1 \quad (5.95)$$

$$= 0, \quad (5.96)$$

Dove (a) segue dal bound su  $\operatorname{sgn}(x)$  e (b) segue dal fatto che  $l'(x)$  soddisfi la Kraft inequality.

Abbiamo l'uguaglianza nella catena qui sopra solo se abbiamo l'uguaglianza in (a) e (b).

Abbiamo l'uguaglianza nel bound di  $\operatorname{sgn}(t)$  se  $t = 0$  o  $t = 1$ .

L'uguaglianza in (b) implica che  $l'(x)$  soddisfi la K.i. con l'uguaglianza.

Combinare questi due fatti implica che  $l'(x) = l(x) \forall x$ .

## Arithmetic Coding

La procedura di Huffman descritta in precedenza è ottimale per fare encoding di una variabile random con una **distribuzione conosciuta** che deve essere encoded simbolo per simbolo. Però, dato il fatto che le lunghezze delle codewords per un Huffman code sono state ristrette per essere **intere**, ci potrebbe essere una loss fino a 1 bit per simbolo nell'efficienza del coding. Per ovviare a questo problema utilizziamo l'arithmetic coding: invece di usare una sequenza di bit per rappresentare un simbolo, lo rappresentiamo utilizzando un **sotto-intervallo** dell'intervallo dell'unità.

Il codice per una sequenza di simboli sarà dunque un intervallo le cui lunghezze decrescono nell'aggiungere simboli alla sequenza. Questa proprietà ci permette di avere un coding scheme che è **incrementale** e per il quale le lunghezze delle codewords non sono costrette ad essere integrali.

## Arithmetic Coding for dummies

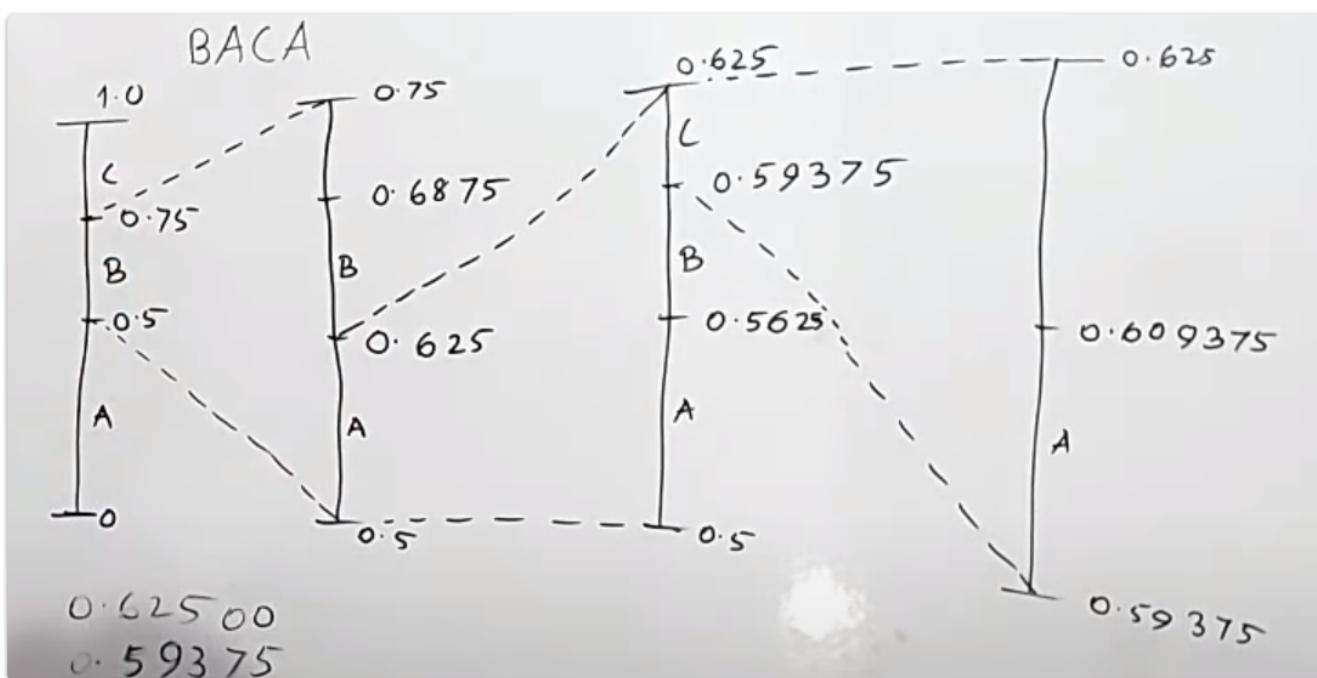
La differenza tra Huffman e Arithmetic coding è che invece di separare l'input in simboli e rimpiazzare ognuno di questi con un codice, l'arithmetic coding fa encoding dell'intero messaggio in **un solo numero**, una frazione  $q$  a precisione arbitraria, compresa tra 0.0 e 1.0. Esso rappresenta l'informazione corrente come un range, definito da due numeri.

Immaginiamo  $P(A) = 0.5$     $P(B) = 0.25$     $P(C) = 0.25$  e il codice *BACA* sull'alfabeto  $\chi = A, B, C$ .

Allora, per l'A.C.:

$$\begin{aligned} A &\in [0, 0.5) \\ B &\in [0.5, 0.75) \\ C &\in [0.75, 1] \end{aligned}$$

- Il primo carattere si tratta di *B*, quindi facciamo zoom su quella parte, dove il nostro nuovo limite generale diventa tra 0.5 e 0.75.
- Per ottenere il secondo carattere, *A*, dobbiamo prendere il primo 50%, dunque il calcolo:  $(0.75 - 0.5) \cdot 0.5 + 0.5 = 0.625$ . Ora siamo tra 0.5 e 0.625.
- Per *C*:  $75\% \cdot (0.625 - 0.5) + 0.5 = 0.59375$ . Ora siamo tra 0.59375 e 0.625
- Per *A*:  $50\% \cdot (0.625 - 0.59375) + 0.59375 = 0.609375$ . Quindi, intervallo finale: [0.59375, 0.609375].



[Un semplice esempio di A.C.](#)

Supponiamo di avere una sequenza di simboli, che provengono da un alfabeto con tre elementi:  $A, B, C$ . Un semplice codificatore a blocchi potrebbe trasformare ogni simbolo in una sequenza di due bit, ma sarebbe uno spreco, due bit possono esprimere quattro combinazioni e quindi una combinazione non verrebbe mai usata.

Possiamo pensare di rappresentare i simboli come un numero razionale compreso tra 0 e 1 in base 3 e di rappresentare ogni simbolo come una cifra del numero. Per esempio la sequenza  $ABBCAB$  verrebbe convertita in 0.0112013. Questa poi potrebbe essere convertita in base due e potrebbe diventare per esempio 0.0010110012 — questa sequenza usa 9 bit e al posto dei 12 bit richiesti da un codificatore ingenuo e quindi occupa il 25% in meno. Il decodificatore ovviamente dovrebbe fare i passi opposti per ottenere la sequenza di partenza.

## Encoding

Per esempio, un semplice modello statico potrebbe definire:

- 60% di possibilità di avere un simbolo NEUTRO;
- 20% di possibilità di avere un simbolo POSITIVO;
- 10% di possibilità di avere un simbolo NEGATIVO;
- 10% di possibilità di avere un simbolo di FINE-DEL-FILE (questo simbolo serve al decodificatore per dire che il flusso da decodificare è terminato).

Suddividiamo gli intervalli tramite una probabilità fissa.

- Il simbolo NEUTRO ha intervallo [0, 0.6)
- Il simbolo POSITIVO ha intervallo [0.6, 0.8)
- Il simbolo NEGATIVO ha intervallo [0.8, 0.9)
- Il simbolo FINE-DEL-FILE ha intervallo [0.9, 1)

Quando tutti i simboli vengono codificati il risultato è un intervallo che identifica in modo non ambiguo la sequenza di simboli che l'ha generato. Quindi ogni sistema che conosce il modello **e** l'intervallo è in grado di ricostruire la sequenza di simboli.

## Decoding (opzionale)

Supponiamo di dover decodificare la serie di simboli associati al valore 0.538, si inizia la decodifica partendo con un intervallo [0, 1) e utilizzando il modello indicato sopra si divide l'intervallo in quattro sotto-intervalli. Il primo simbolo secondo il nostro modello è NEUTRO dato che il numero vale meno di 0.6 e quindi ricade nell'intervallo NEUTRO.

Adesso dividiamo l'intervallo [0, 0.6) in sotto-intervalli secondo le probabilità indicate all'inizio e otteniamo:

- Il simbolo NEUTRO ha intervallo [0, 0.36) "60% dell'intervallo [0, 0.6)"
- Il simbolo POSITIVO ha intervallo [0.36, 0.48) "20% dell'intervallo [0, 0.6)"
- Il simbolo NEGATIVO ha intervallo [0.48, 0.54) "10% dell'intervallo [0, 0.6)"

- Il simbolo FINE-DEL-FILE ha intervallo  $[0.54, 0.6)$  "10% dell'intervallo  $[0, 0.6)$ "

Il numero 0.538 ricade nell'intervallo  $[0.48, 0.54)$  e quindi il secondo simbolo è NEGATIVO.

Ripetiamo nuovamente la suddivisione del nuovo intervallo e otteniamo:

- Il simbolo NEUTRO ha intervallo  $[0.48, 0.516)$  "60% dell'intervallo  $[0.48, 0.54)$ "
- Il simbolo POSITIVO ha intervallo  $[0.516, 0.528)$  "20% dell'intervallo  $[0.48, 0.54)$ "
- Il simbolo NEGATIVO ha intervallo  $[0.528, 0.534)$  "10% dell'intervallo  $[0.48, 0.54)$ "
- Il simbolo FINE-DEL-FILE ha intervallo  $[0.534, 0.540)$  "10% dell'intervallo  $[0.48, 0.54)$ "

Il numero 0.538 ricade entro l'intervallo  $[0.535, 0.54)$  che appartiene al simbolo FINE-DEL-FILE quindi abbiamo raggiunto la fine del flusso di simboli da decodificare. È da notare che anche i numeri 0.534, 0.535, 0.536, 0.537, 0.539 sarebbero andati comunque bene. Questo potrebbe far pensare che il sistema soffra di alcune inefficienze, in realtà questo si ottiene perché nell'esempio abbiamo lavorato in base decimale, convertendo il tutto in base binaria avremmo usato come numero di decodifica 0.10001010 (che in decimale vale 0.5390625) che costa 8 bit.

La nostra codifica occupa 8 bit che è meglio dei 12 bit utilizzati da un codificatore banale ma comunque sono molti più bit di quelli definiti a livello teorico. Secondo la teoria dell'entropia ogni simbolo dovrebbe occupare 1.57 bit che moltiplicato per i tre simboli trasmessi fa 4.71 bit. Questa differenza tra il miglior risultato a livello teorico e il risultato del codificatore è dovuto al numero ridotto di simboli codificati, all'aumento del numero di simboli da codificare la differenza con il risultato teorico si assottiglia notevolmente. Inoltre le probabilità indicate nell'esempio sono molto diverse da quelle presenti nel messaggio codificato. Le reali probabilità del messaggio sono  $[0.33, 0, 0.33, 0.33]$ , usando queste probabilità gli intervalli sarebbero  $[0, 1/3]; [1/9, 2/9]; [5/27, 6/27]$ ; in binario diverrebbero  $[1011110, 1110001]$  e quindi si potrebbe inviare che solamente il valore 111 per indicare la sequenza di simboli. Questo mostra anche che una statistica sbagliata può peggiorare significativamente le prestazioni della codifica.

Le motivazioni per l'arithmetic coding derivano dal *SEF coding* e dal seguente lemma:

### Lemma 13.3.1

Sia  $Y$  una variabile random con una funzione di distribuzione di probabilità continua  $F(Y)$ . Sia  $U = F(Y)$ , allora  $U$  è uniformemente distribuita su  $[0, 1]$ .

*Proof:* visto che  $F(Y) \in [0, 1]$ , il range di  $U$  va da 0 ad 1 compreso. Inoltre,

$$\begin{aligned} F_U(u) &= \Pr(U \leq u) \\ &= \Pr(F(Y) \leq u) \\ &= \Pr(Y \leq F^{-1}(u)) \\ &= F(F^{-1}(u)) \\ &= u, \end{aligned}$$

Che dimostra che  $U$  ha una distribuzione uniforme su  $[0, 1]$ ,  $\square$ .

Ora consideriamo una sequenza infinita di variabili random  $X_1, X_2, \dots$  con un alfabeto finito  $\chi = 0, 1, 2, \dots, m$ . Per ogni sequenza  $x_1, x_2, \dots$ , da questo alfabeto, possiamo piazzare "0." di fronte ai simboli e considerarli numeri reali (con base  $m + 1$ ) tra 0 ed 1. Sia  $X$  la variabile reale random  $X = 0.X_1X_2\dots$ , allora  $X$  ha la seguente funzione di distribuzione:

$$\begin{aligned} F_X(x) &= \Pr(X \leq x = 0.x_1x_2\dots) \\ &= \Pr(0.X_1X_2\dots \leq 0.x_1x_2\dots) \\ &= \Pr(X_1 < x_1) + \Pr(X_1 = x_1, X_2 < x_2) + \dots \end{aligned}$$

Sia dunque ora  $U = F_X(X) = F_X(0.X_1X_2\dots) = 0.F_1F_2\dots$ . Se la distribuzione sulle infinite sequenza  $X^\infty$  non ha atomi, allora, per il lemma qui sopra,  $U$  ha una distribuzione uniforme su  $[0, 1]$  e dunque i bit  $F_1F_2\dots$  nell'espansione binaria di  $U$  sono *Bernoulli*( $\frac{1}{2}$ ). Questi bit sono perciò non comprimibili e formano una rappresentazione *compressa* della sequenza  $0.X_1X_2\dots$ . Per modelli di Markov o Bernoulli, è facile calcolare la funzione di distribuzione cumulativa, come andremo a vedere nel prossimo esempio.

### Esempio 13.3.1

Sia  $X_1, X_2, \dots, X_n$  una *Bernoulli*( $p$ ). Allora la sequenza  $x^n = 110101$  si mappa in:

$$\begin{aligned} F(x^n) &= \Pr(X_1 < 1) + \Pr(X_1 = 1, X_2 < 1) \\ &\quad + \Pr(X_1 = 1, X_2 = 1, X_3 < 0) \\ &\quad + \Pr(X_1 = 1, X_2 = 1, X_3 = 0, X_4 < 1) \\ &\quad + \Pr(X_1 = 1, X_2 = 1, X_3 = 0, X_4 = 1, X_5 < 0) \\ &\quad + \Pr(X_1 = 1, X_2 = 1, X_3 = 0, X_4 = 1, X_5 = 0, X_6 < 1) \end{aligned} \tag{13.56}$$

$$= q + pq + p^2 \cdot 0 + p^2 q \cdot q + p^2 q p \cdot 0 + p^2 q p q q \tag{13.57}$$

$$= q + pq + p^2 q^2 + p^3 q^3. \tag{13.58}$$

Da notare come ogni termine sia facilmente calcolabile dal termine prima. In generale, per un processo binario arbitrario  $\{X_i\}$ ,

$$F(x^n) = \sum_{k=1}^n p(x^{k-1}0)x_k.$$

La trasformata della probabilità quindi forma un mapping da delle sequenze sorgente infinite a delle sequenze binarie infinite **non-comprimibili**.

Guardiamo però ora quanto abbiamo compresso da questa trasformata su sequenze

finite. Sia  $X_1, X_2, \dots, X_n$  una sequenza di variabili random binarie di lunghezza  $n$ , e sia  $x_1, x_2, \dots, x_n$  un particolare outcome. Possiamo trattare questa sequenza come rappresentativa di un intervallo  $[0.x_1x_2\dots x_n000\dots, 0.x_1x_2\dots x_n1111\dots]$ . Questo è il set di sequenze infinite che partono con  $0.x_1x_2\dots x_n$ .

Sotto questa trasformata della probabilità, questo intervallo viene mappato in un altro intervallo,  $[F_Y(0.x_1x_2\dots x_n), F_Y(0.x_1x_2\dots x_n + (1/2)^n)]$ , le quali lunghezze sono uguali a  $P_X(x_1, x_2, \dots, x_n)$ , la somma delle probabilità di tutte le sequenze infinite che partono con  $0.x_1x_2\dots x_n$ . Sotto la trasformata inversa della probabilità, ogni numero reale  $u$  dentro questo intervallo viene mappato in una sequenza che parte con  $x_1, x_2, \dots, x_n$ , e quindi dati  $u$  ed  $n$ , possiamo ricostruire  $x_1, x_2, \dots, x_n$ . Il SEF Coding ci permette di costruire un prefix-free code di lunghezza  $\log \frac{1}{p(x_1, x_2, \dots, x_n)} + 2$  bits, e quindi è possibile fare encoding della sequenza  $x_1, \dots, x_n$  con questa lunghezza.

**NB:**  $\log \frac{1}{p(x_1, x_2, \dots, x_n)}$  è la lunghezza ideale della codeword per  $x^n$

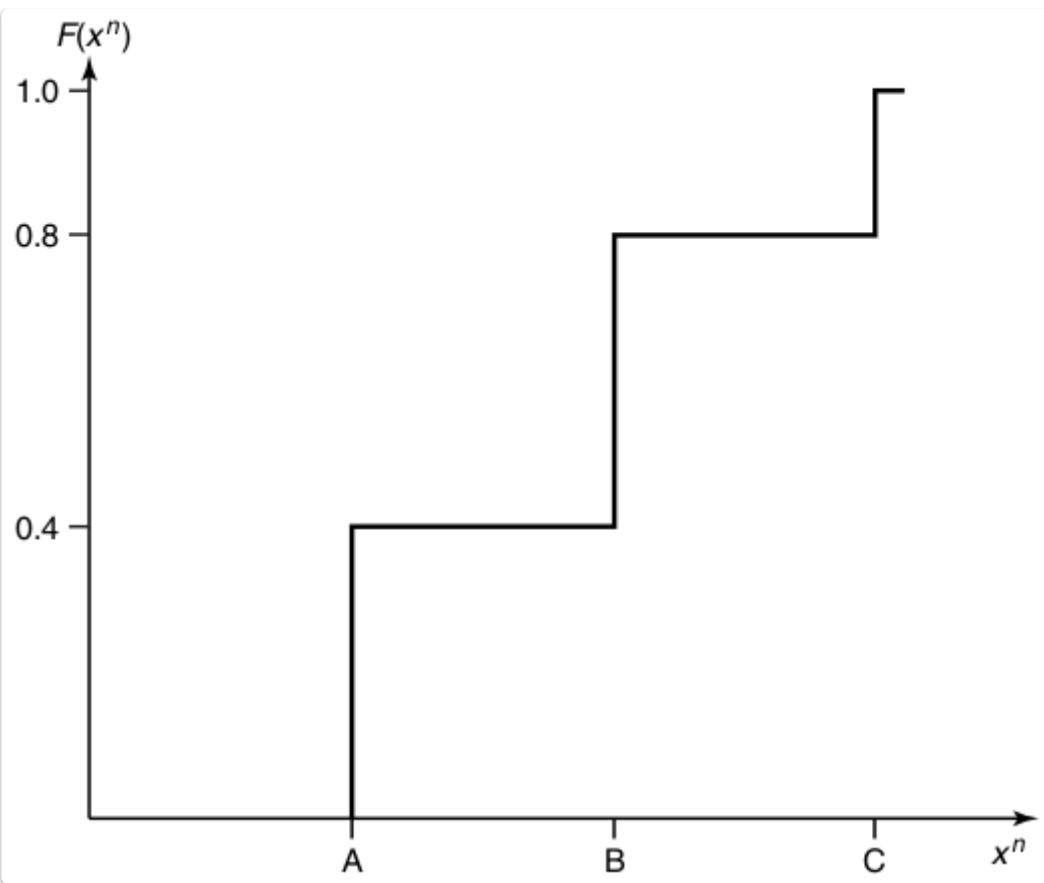
Il processo di encoding della sequenza con la funzione di distribuzione cumulativa così descritta assume un'accuratezza arbitraria per la computazione. In pratica, quindi, dobbiamo implementare tutti i numeri con una precisione finita, e descriverli con un'implementazione. La chiave non è considerare punti di infinita-precisione per la funzione di distribuzione cumulativa ma intervalli nell'intervallo di unità.

Ogni sequenza di lunghezza finita di simboli può corrispondere ad un sotto-intervallo dell'intervallo dell'unità.

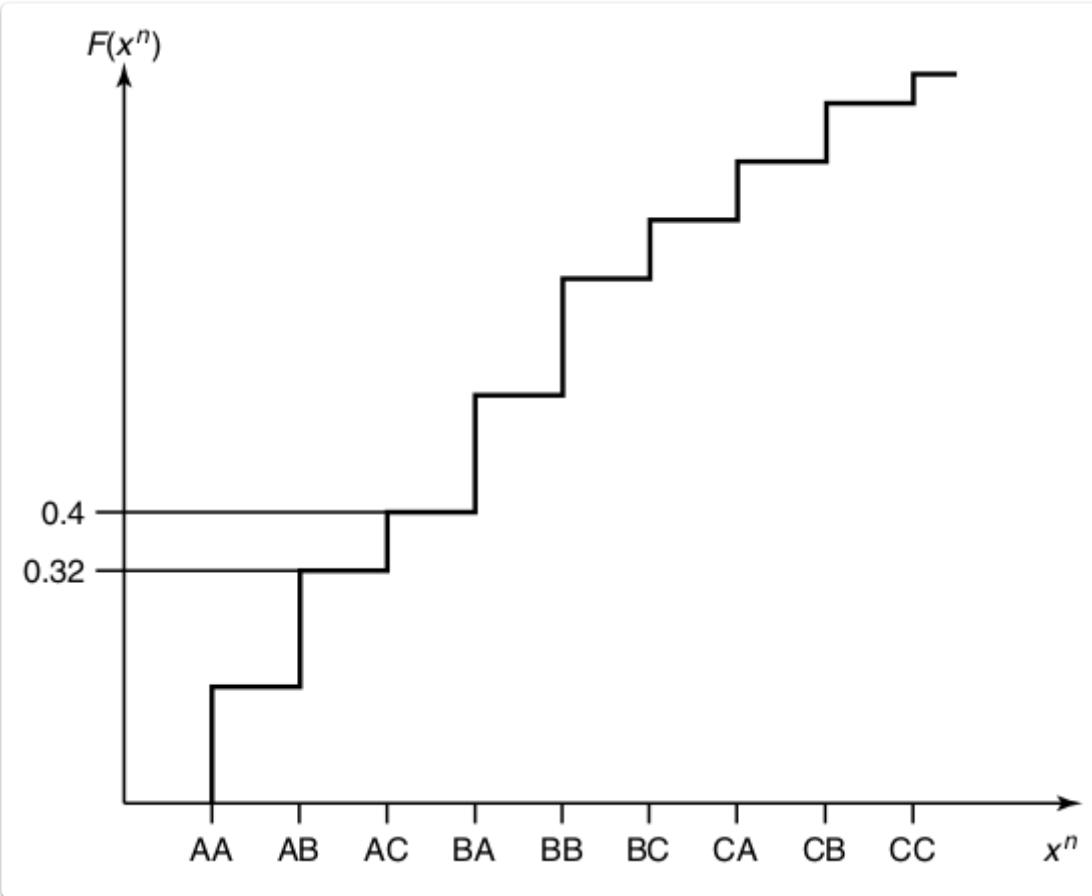
L'obiettivo dell'arithmetic encoding è di rappresentare una sequenza di variabili random in un sotto-intervallo  $[0, 1]$ . All'aumentare dei simboli in input, la lunghezza del sotto-intervallo che corrisponde alla sequenza in input decresce. Vediamo un esempio:

### Esempio 13.3.2 (arithmetic coding per un alfabeto input ternario)

Consideriamo una variabile  $X$  con un alfabeto ternario  $\{A, B, C\}$ , che hanno probabilità rispettivamente di 0.4, 0.4 e 0.2. Sia la sequenza su cui fare encoding  $ACAA$ . Quindi,  $F_l(\cdot) = (0, 0.4, 0.8)$  e  $F_h(\cdot) = (0.4, 0.8, 1.0)$ . Inizialmente, la sequenza input è vuota, e l'intervalllo corrispondente è dunque  $[0, 1)$ . La funzione di distribuzione cumulativa dopo il primo simbolo in input  $A$  è mostrata qui:



Viene facile calcolare che l'intervallo nell'algoritmo senza scalare dopo il primo simbolo è  $[0, 0.4]$ ; dopo il secondo simbolo,  $C$ , sarà  $[0.32, 0.4]$ :



Dopo il terzo simbolo,  $A$ , sarà  $[0.32, 0.352]$  e dopo il quarto ed ultimo simbolo,  $A$ , sarà  $[0.32, 0.3328]$ . Visto che la probabilità di questa sequenza è 0.0128, dobbiamo utilizzare

$\log(\frac{1}{0.0128}) + 2$  bits (quindi 9 bits se facciamo ceiling) per fare encoding del *midpoint* della sequenza di intervallo utilizzando SEF coding (0.3264, che è 0.010100111 binario).

## Sommario Arithmetic Coding

La A.C., data una lunghezza  $n$  e una funzione di probabilità di massa  $q(x_1, x_2, \dots, x_n)$  ci  
abilità a fare encoding delle sequenze  $x_1x_2\dots x_n$  in un codice di lunghezza  
 $\log \frac{1}{q(x_1x_2\dots x_n)} + 2$  bits. Se la sorgente è i.i.d. e la distribuzione assunta  $q$  è uguale alla  
distribuzione  $p$  dei dati in input, la procedura riesce ad arrivare intorno a 2 bits  
dall'entropia. La procedura è inoltre incrementale e può essere usata per ogni  
lunghezza di blocco.

# Asymptotic Equipartition Property

In teoria dell'informazione l'analogo della teoria dei grandi numeri è l'Asymptotic Equipartition Property (che da ora in poi scriverò come **AEP**). Questa è la diretta conseguenza della legge *debole* dei grandi numeri.

*La legge dei grandi numeri dice che per delle variabili i.i.d. (indipendenti, identicamente distribuite),  $\frac{1}{n} \sum_{i=1}^n X_i$  è vicina al valore atteso  $EX$  per valori grandi di  $n$ .*

La AEP invece ci dice una cosa molto simile, ovvero che  $\frac{1}{n} \log \frac{1}{p(X_1, X_2, \dots, X_n)}$  è vicina all'entropia  $H$ , dove le  $X$  sono variabili random e  $p(\dots)$  è la probabilità di osservare quella sequenza. Quindi, per tirare le somme, la probabilità  $p(\dots)$  assegnata ad una sequenza osservata sarà vicina a  $2^{-nH}$ .

Questo ci permette di dividere il set di tutte le sequenze in due gruppi:

- Il **typical set**: dove l'entropia dell'esempio estratto è vicina all'entropia vera
- Il **non-typical set**: che contiene tutto il resto delle sequenze  
Ovviamente noi ci concentreremo sul primo set.

## Primo esempio

Sia una variabile  $X \in \{0, 1\}$  che ha una funzione di probabilità di massa definita come  $p(1) = p, p(0) = q$ . Se  $X_1, X_2, \dots, X_n$  sono i.i.d. allora per  $p(x)$  la probabilità della sequenza  $x_1, x_2, \dots, x_n$  sarà  $\prod_{i=1}^n p(x_i)$ . Per esempio, la probabilità della sequenza  $1, 0, 1, 1, 0, 1$  è  $p^{\sum X_i} q^{n - \sum X_i} = p^4 q^2$ .

Chiaramente, non è vero che tutte le  $2^n$  sequenze di lunghezza  $n$  hanno la stessa probabilità!

In ogni caso, possiamo predire la probabilità della sequenze che andremo ad osservare. Noi chiediamo la probabilità  $p(X_1, X_2, \dots, X_n)$  degli outcomes  $X_{\dots n}$ , dove tutti questi  $X$  sono i.i.d.. Apparentemente, stiamo chiedendo la probabilità di un evento preso dalla stessa identica distribuzione di probabilità.

*Potremmo riassumere questo fatto con la frase "Quasi tutti gli eventi sono egualmente sorprendenti"*

E che potremmo formalizzare come:

$$Pr\{(X_1, X_2, \dots, X_n) : p(X_1, X_2, \dots, X_n) = 2^{-n(H \pm \epsilon)}\} \approx 1$$

Nell'esempio appena dato, diciamo semplicemente che il numero di 1 nella sequenza è vicino a  $np$  con alta probabilità e che tutte le sequenze hanno più o meno la stessa

probabilità  $2^{-nH(p)}$ . Da notare l'utilizzo dell'idea di convergenza della probabilità, definita qui nell'immagine seguente:

**Definition** (*Convergence of random variables*). Given a sequence of random variables,  $X_1, X_2, \dots$ , we say that the sequence  $X_1, X_2, \dots$  converges to a random variable  $X$ :

1. *In probability* if for every  $\epsilon > 0$ ,  $\Pr\{|X_n - X| > \epsilon\} \rightarrow 0$
2. *In mean square* if  $E(X_n - X)^2 \rightarrow 0$
3. *With probability 1* (also called *almost surely*) if  $\Pr\{\lim_{n \rightarrow \infty} X_n = X\} = 1$

## Teorema AEP

Se  $X_1, X_2, \dots$  sono i.i.d.  $\sim p(x)$ , allora:

$$-\frac{1}{n} \log p(X_1, X_2, \dots, X_n) \rightarrow H(X)$$

in probabilità.

*Proof:* funzioni di variabili indipendenti sono anch'esse variabili indipendenti random. Quindi, visto che  $X_i$  sono i.i.d., così anche  $\log p(X_i)$ . Quindi, per la legge debole dei grandi numeri,

$$\begin{aligned} -\frac{1}{n} \log p(X_1, X_2, \dots, X_n) &= -\frac{1}{n} \sum_i \log p(X_i) \\ &\rightarrow -E \log p(X) \text{ in probability} \\ &= H(X), \end{aligned}$$

Che prova il teorema,  $\square$ .

## Typical Set

Un set tipico  $A_\epsilon^{(n)}$  rispetto a  $p(x)$  è il set di sequenze  $x_1, x_2, \dots, x_n \in \chi$  con la seguente proprietà:

$$2^{-n(H(X)+\epsilon)} \leq p(x_1, x_2, \dots, x_n) \leq 2^{-n(H(X)-\epsilon)}$$

Come conseguenza dell'AEP, possiamo mostrare che il set  $A_\epsilon^{(n)}$  ha le seguenti proprietà:

1. Se  $(x_1, x_2, \dots, x_n) \in A_\epsilon^{(n)}$ , allora  $H(X) - \epsilon \leq -\frac{1}{n} \log p(x_1, x_2, \dots, x_n) \leq H(X) + \epsilon$ .
2.  $\Pr(A_\epsilon^{(n)}) > 1 - \epsilon$  per  $n$  abbastanza grande
3.  $|A_\epsilon^{(n)}| \leq 2^{n(H(X)+\epsilon)}$ , dove  $|A|$  denota il numero di elementi nel set  $A$ .
4.  $|A_\epsilon^{(n)}| \geq (1 - \epsilon)2^{n(H(X)-\epsilon)}$  per  $n$  abbastanza grande

Quindi, il typical set ha probabilità vicina ad 1 e tutti gli elementi sono quasi equiprobabili, e il numero degli elementi del typical set è vicino a  $2^{nH}$ .

*Proof:* la prova della proprietà (1) è immediata dalla definizione di A. La prova della proprietà (2) deriva direttamente dal teorema AEP: visto che la probabilità di un evento tende ad 1 quando n tende ad infinito, quindi, per ogni  $\delta > 0$  esiste un  $n_0$  tale che  $\forall n \geq n_0$ :

$$Pr\left(\left| -\frac{1}{n} \log p(X_1, X_2, \dots, X_n) - H(X) \right| < \epsilon\right) > 1 - \delta$$

Se mettiamo  $\delta = \epsilon$  otteniamo la seconda parte del teorema.

Per provare (3), scriviamo:

$$\begin{aligned} 1 &= \sum_{x \in \chi^n} p(x) \\ &\geq \sum_{x \in A_\epsilon^{(n)}} p(x) \\ &\geq \sum_{x \in A_\epsilon^{(n)}} 2^{-n(H(X)+\epsilon)} \\ &= 2^{-n(H(X)+\epsilon)} |A_\epsilon^{(n)}| \end{aligned}$$

Dove la seconda diseguaglianza segue dalla formula originale del typical set. Poi:

$$|A_\epsilon^{(n)}| \leq 2^{n(H(X)+\epsilon)}.$$

E per l'ultima (4) possiamo dimostrarla come segue: per un  $n$  sufficientemente grande,  $Pr(2^{-n(H(X)+\epsilon)}) > 1 - \epsilon$ , e allora:

$$\begin{aligned} 1 - \epsilon &< Pr(A_\epsilon^{(n)}) \\ &\leq \sum_{x \in A_\epsilon^{(n)}} 2^{-n(H(X)+\epsilon)} \\ &= 2^{-n(H(X)+\epsilon)} |A_\epsilon^{(n)}| \end{aligned}$$

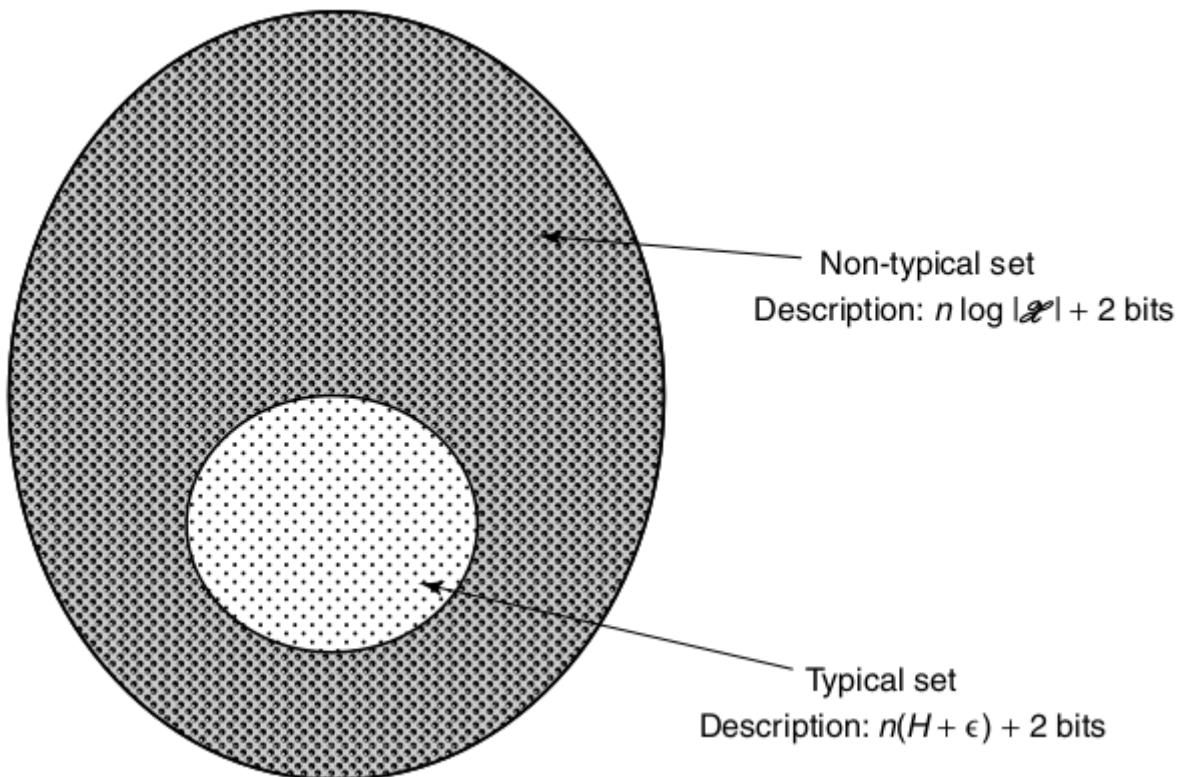
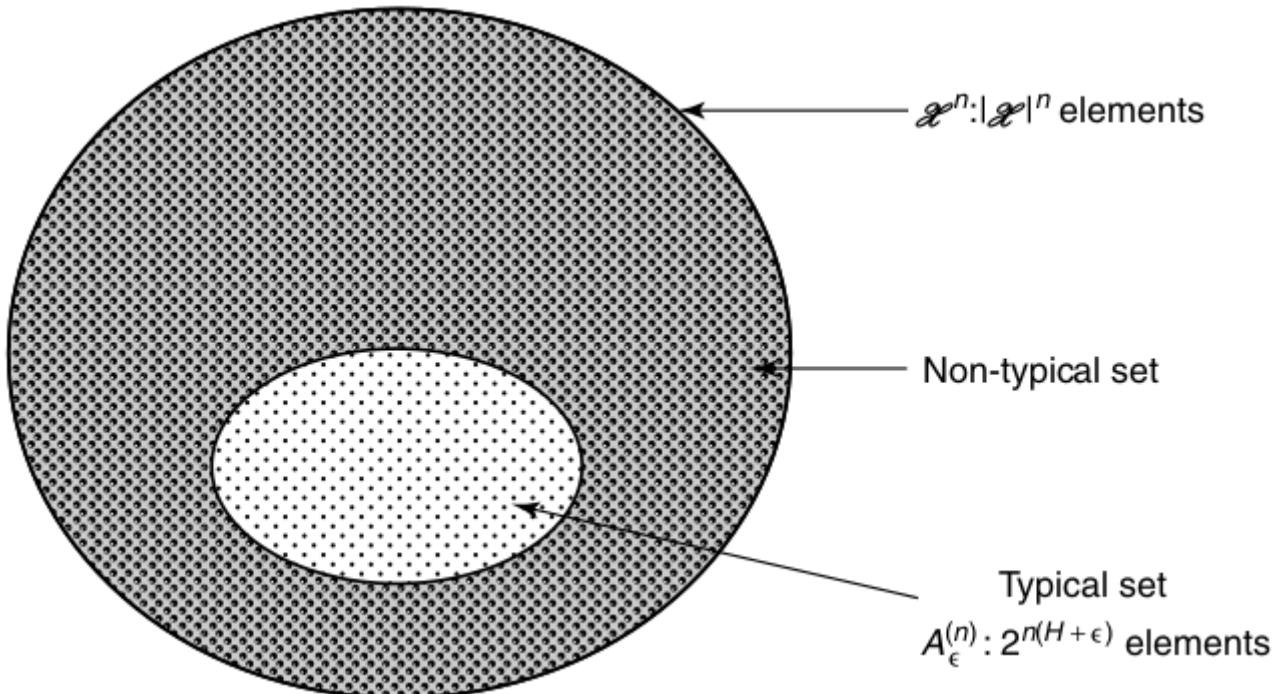
E quindi:

$$|A_\epsilon^{(n)}| \geq (1 - \epsilon) 2^{n(H(X)+\epsilon)}$$

□.

## Conseguenze dell'AEP: data compression

Sia  $X_1, X_2, \dots, X_n$  variabili i.i.d. prese da una p.m.f.  $p(x)$ . Vorremmo trovare descrizioni corte per queste sequenze di variabili random. Dividiamo dunque tutte le sequenze di  $\chi^n$  in 2 sets: il typical set  $A_\epsilon^{(n)}$  e il suo complementare, come ora mostriamo in figura.



Ordiniamo gli elementi in ogni set secondo l'ordine lessicografico, quindi possiamo rappresentare ogni sequenza del typical set dando l'indice della sequenza del set. Visto che ci sono  $\leq 2^{n(H+\epsilon)}$  sequenze nel typical set, l'indice non richiede più di  $n(H + \epsilon) + 1$  bits. Mettiamo uno 0 prefisso davanti ad ognuna, rendendo la lunghezza massima di  $n(H + \epsilon) + 2$  bits. Similmente, possiamo indexare ogni sequenza nel non-typical set utilizzando non più di  $n \log |\chi| + 1$  bits. Mettiamo un 1 prefisso a queste, ottenendo un codice per ogni sequenza in  $\chi^n$ .

Abbiamo così creato un coding scheme dalle seguenti proprietà:

- Il codice è 1-a-1 e facilmente decodabile. Il bit iniziale è una flag per dirci la lunghezza della codeword che segue.
- Abbiamo utilizzato un'enumerazione brute-force del set non-typical senza prendere conto del fatto che il numero di elementi nel non-typical set è minore degli elementi in  $\chi^n$ . Sorprendentemente, questo è bastevole per dare una descrizione efficiente.
- Le sequenze tipiche hanno una descrizione breve di lunghezza che si avvicina a  $nH$ .

Utilizziamo una notazione  $x^n$  per denotare una sequenza  $x_1, x_2, \dots, x_n$ . Sia  $l(x^n)$  la lunghezza della codeword che corrisponde a  $x^n$ . Se  $n$  è sufficientemente grande così che  $Pr(A_\epsilon^{(n)}) \geq 1 - \epsilon$ , la lunghezza attesa della codeword sarà:

$$\begin{aligned}
E(l(X^n)) &= \sum_{x^n} p(x^n)l(x^n) \\
&= \sum_{x^n \in A_\epsilon^{(n)}} p(x^n)l(x^n) + \sum_{x^n \in A_\epsilon^{(n)c}} p(x^n)l(x^n) \\
&\leq \sum_{x^n \in A_\epsilon^{(n)}} p(x^n)(n(H + \epsilon) + 2) + \sum_{x^n \in A_\epsilon^{(n)c}} p(x^n)(n \log |\chi| + 2) \\
&= Pr(A_\epsilon^{(n)})n(H + \epsilon + 2) + Pr(A_\epsilon^{(n)c})(n \log |\chi| + 2) \\
&\leq n(H + \epsilon) + \epsilon n(\log |\chi|) + 2 \\
&= n(H + \epsilon')
\end{aligned}$$

Dove  $\epsilon' = \epsilon + \epsilon \log |\chi| + \frac{2}{n}$  che può esser reso arbitrariamente piccolo da una scelta appropriata di  $\epsilon$  seguita da una scelta appropriata di  $n$ . Dunque abbiamo provato il teorema seguente.

### Teorema 3.2.1

Sia  $X^n$  i.i.d.  $\sim p(x)$ . Sia  $\epsilon > 0$ . Allora esiste un codice che mappa sequenze  $x^n$  di lunghezza  $n$  in stringhe binarie tali che il mapping sia 1-a-1, ovvero invertibile, e:

$$E\left[\frac{1}{n}l(X^n)\right] \leq H(X) + \epsilon$$

per  $n$  sufficientemente grande.

E quindi possiamo rappresentare sequenze  $X^n$  utilizzando  $nH(X)$  bit in media.

### AEP for dummies

---

Il teorema ci dice che anche se un processo random produce molte serie di risultati, quella che viene poi veramente prodotta è molto probabilmente estratta da un set abbastanza definito di outcomes che hanno tutti approssimativamente la stessa chance di essere estratti.

Nonostante perciò ci siano degli outcome individuali che in se hanno probabilità più alta di essere estratti, a lungo andare nell'estrazione i risultati verranno comunque dal set tipico.

# Esercizi

## Esercizio misto

**N. 2** — A discrete memoryless source emits a sequence of statistically independent binary digits with probability  $p(1) = 0.01$  and  $p(0) = 0.99$ . The digits are taken 100 at a time and a binary codeword is provided for every sequence of 100 digits containing three or fewer ones.

1. Assuming that all codewords are the same length, find the minimum length required to provide codewords for all sequences with three or fewer ones.
2. Compare the minimum length of the code derived in the previous step with the source entropy  $H(X)$ . What kind of consideration this comparison suggests ?
3. Calculate the probability of observing a source sequence for which no codeword has been assigned.

1. Qui ci viene chiesto di trovare la lunghezza minima delle codewords per tutte quelle codewords con 3 o meno "1". Per farlo, utilizziamo una binomiale come segue:  $\binom{100}{0} + \binom{100}{1} + \binom{100}{2} + \binom{100}{3} = 166751$  (che tradotto sarebbe, tutte le sequenze con 0 "1", + tutte le sequenze con 1 "1"... etc...). Dunque, la lunghezza minima sarebbe:  $n = \lceil \log_2 166751 \rceil = 18$ .
2.  $H(X) = -0.01 \cdot \log_2 0.01 - 0.99 \cdot \log_2 0.99 = 0.08079\dots$  Applicando i risultati dell'A.E.P. scopriamo che riusciamo a ridurre la lunghezza minima dei codici a:  
 $l = nH(X) = 100 \cdot 0.08079 \approx 9\text{bits}$ , molto meno di prima!
3. Qua ci chiede di calcolare la

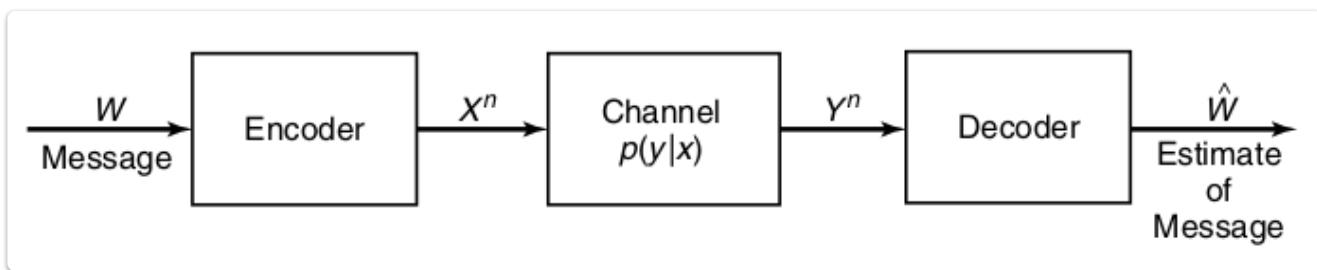
$$P(\text{piu di 3 "1" nella codeword}) = 1 - P(\text{meno o uguale a 3 "1" nella codeword}) =$$

$$1 - \sum_{k=0}^3 \binom{100}{k} \cdot 0.01^k \cdot 0.99^{100-k} = 0.018374\dots$$

# Channel Capacity

## Discrete memoryless channel

Un canale discreto è un sistema che consiste in un alfabeto  $\chi$  e un alfabeto output  $\Upsilon$  e una matrice di transizione di probabilità  $p(y|x)$  che esprime la probabilità di osservare un simbolo output  $y$  dato che mandiamo il simbolo  $x$ . Il canale è detto *memoryless* in quanto la distribuzione di probabilità dell'output dipende solamente dall'input a quel tempo ed è condizionalmente indipendente dagli input del canale precedente o gli output.



## Capacità

La capacità di un canale di informazione è definita come il massimo valore di informazione mutuale, o:

$$C = \max_{p(x)} I(X; Y)$$

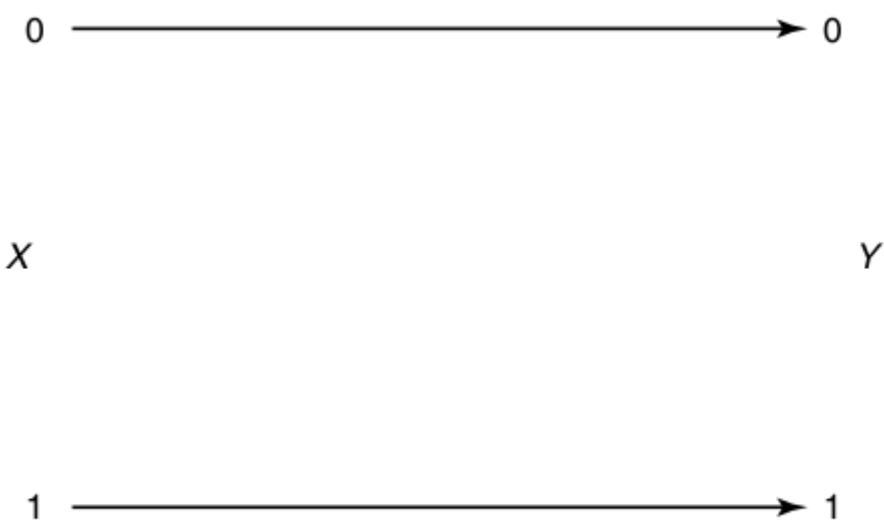
Dove questo massimo è preso su tutte le possibili distribuzioni  $p(x)$ .

Presto scopriremo che la capacità è anche il rateo massimo in bit per channel-flow al quale le informazioni possono essere mandate con probabilità arbitrariamente bassa di errore.

## Esempi di channel capacity

### Canale binario senza rumore

Supponiamo di avere un canale dove l'input binario è riprodotto esattamente all'output, come mostrato in figura, senza errori.

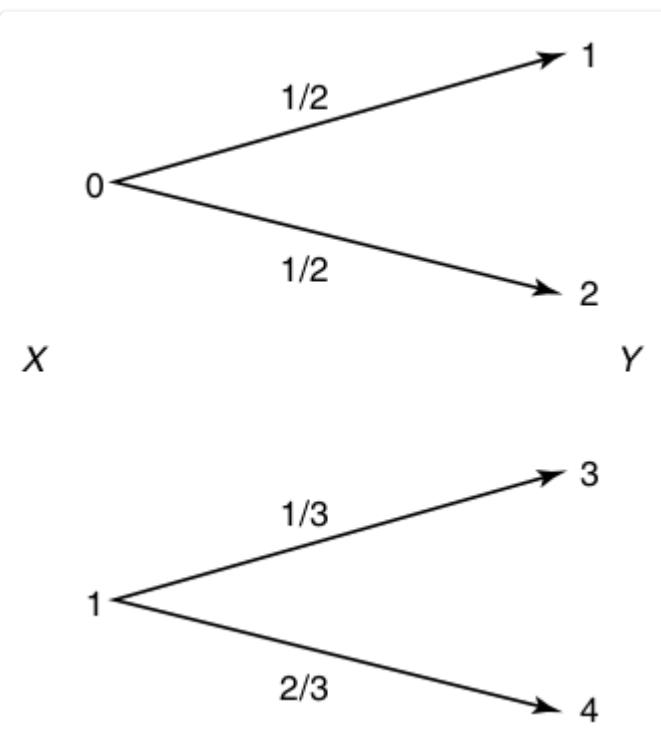


**FIGURE 7.2.** Noiseless binary channel.  $C = 1$  bit.

$$C = \max I(X; Y) = 1 \quad p(x) = (\frac{1}{2}, \frac{1}{2}).$$

### Canale rumoroso con output che non si sovrappongono

Il canale ha due output possibili che corrispondono ai due input (4 outputs totali). Il canale sembra rumoroso, ma non lo è veramente. Anche se l'output del canale è una conseguenza random dell'input, l'input può essere tranquillamente recuperato dall'output, e quindi il bit trasmetto recuperato senza errori. La capacità di canale è quindi 1 bit per trasmissione come prima, con esattamente gli stessi calcoli.

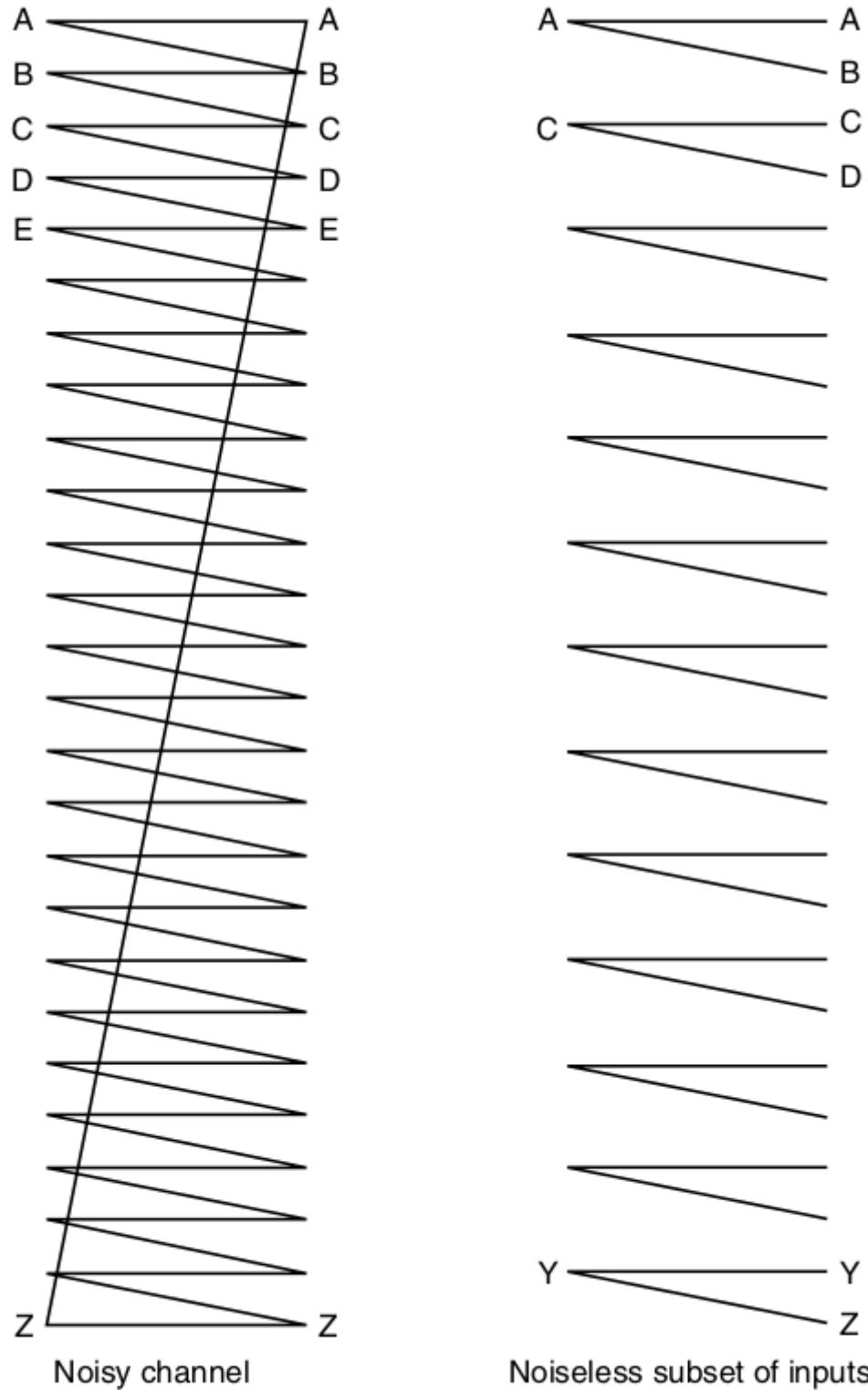


### Macchina da scrivere "rumorosa"

In questo caso il channel input o è ricevuto senza alcun cambio con probabilità  $\frac{1}{2}$  oppure trasformato nella lettera successiva con la stessa probabilità. Se l'input ha 26

simboli e utilizziamo ogni input symbol alternato, possiamo trasmettere uno dei 13 simboli senza errori con ogni trasmissione. Quindi, la capacità del canale è  $\log 13$  bits per trasmissione.

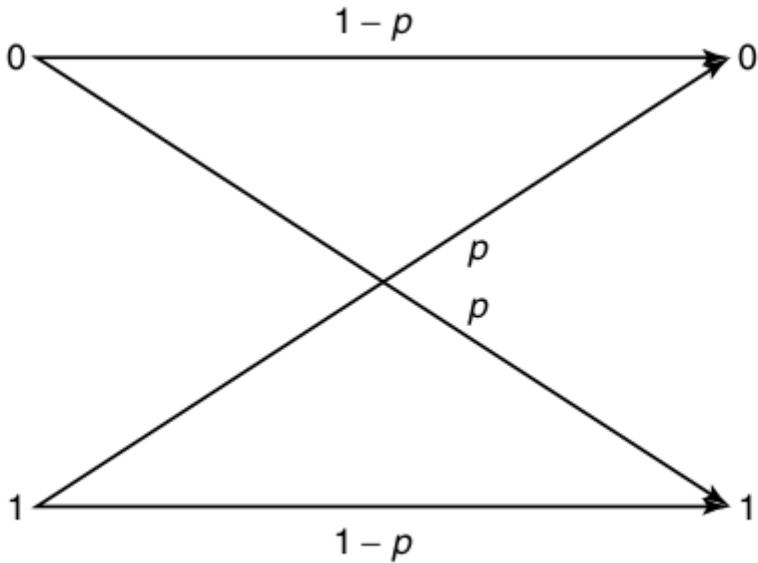
$$C = \max I(X; Y) = \max(H(Y) - H(Y|X)) = \max H(Y) - 1 = \log 26 - 1 = \log 13$$



**FIGURE 7.4.** Noisy Typewriter.  $C = \log 13$  bits.

### Canale simmetrico binario

Consideriamo questo canale simmetrico binario:



**FIGURE 7.5.** Binary symmetric channel.  $C = 1 - H(p)$  bits.

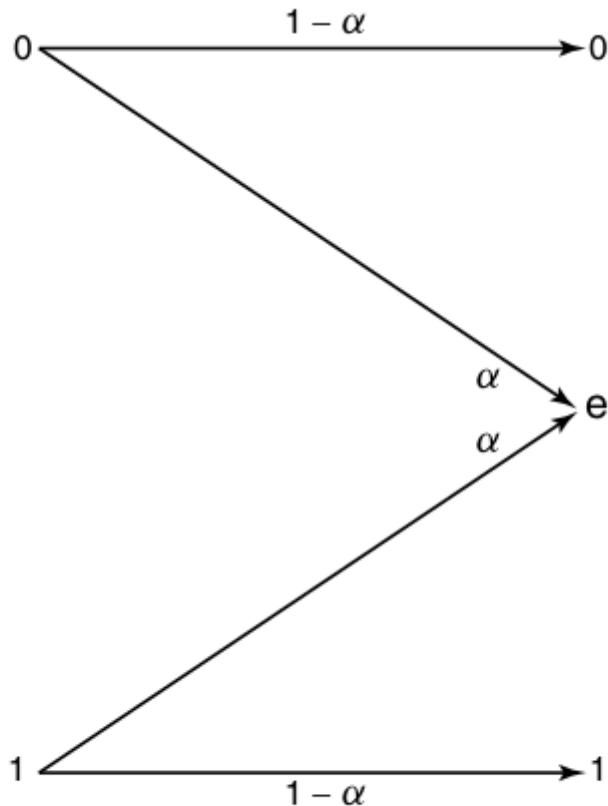
I simboli in input sono complementari con probabilità  $p$ .

Quando avviene un errore, i bit ricevuti non rivelano da dove vengono gli errori. In un senso, tutti i bit ricevuti sono *unreliable*.

$$\begin{aligned}
 I(X; Y) &= H(Y) - H(Y|X) \\
 &= H(Y) - \sum p(x)H(Y|X=x) \\
 &= H(Y) - \sum p(x)H(p) \\
 &= H(Y) - H(p) \\
 &\leq 1 - H(p),
 \end{aligned}$$

Quindi  $C = 1 - H(p)$  bits.

### Binary Erasure Channel



**FIGURE 7.6.** Binary erasure channel.

Analogo di un canale simmetrico binario: invece che essere corrotti, i bit vengono perduti.

Abbiamo 2 input e 3 output. Il ricevente sa quali bit sono stati cancellati.

$$\begin{aligned} C &= \max_{p(x)} I(X; Y) \\ &= \max_{p(x)} (H(Y) - H(Y|X)) \\ &= \max_{p(x)} H(Y) - H(\alpha). \end{aligned}$$

Come primo pensiero viene da dire che  $H(Y) = \log 3$ , ma non potremmo mai arrivare a questo con qualsiasi distribuzione in input  $p(x)$ .

Se scriviamo  $E$  come l'evento  $Y = e$ ,

$$H(Y) = H(Y, E) = H(E) + H(Y|E)$$

E scrivendo  $Pr(X = 1) = \pi$ , abbiamo che:

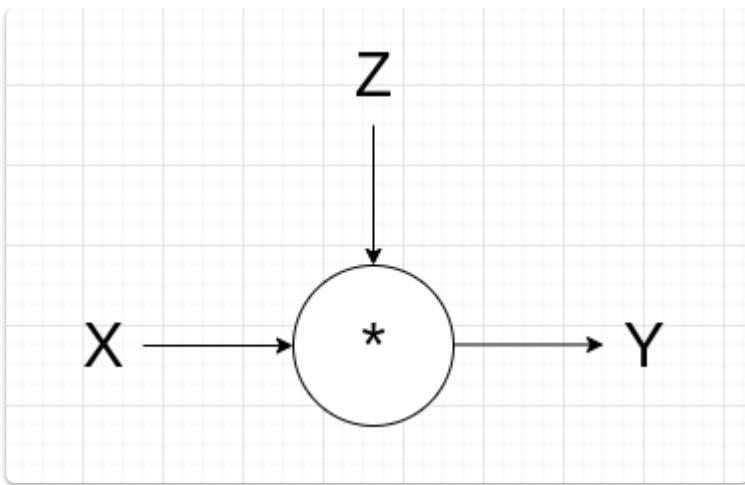
$$H(Y) = H((1 - \pi)(1 - \alpha), \alpha \pi(1 - \alpha)) = H(\alpha) + (1 - \alpha)H(\pi)$$

Quindi sostituendo:

$$C = 1 - \alpha$$

Dove la capacità ha  $\pi = p(x) = \frac{1}{2}$ .

## Binary Multiplicative Channel



Consideriamo un canale moltiplicativo  $Y = XZ$  dove  $X$  e  $Z$  sono variabili aleatorie indipendenti sull'alfabeto  $\{0, 1\}$ . Immaginiamo  $Z$  essere una  $Bernoulli(\alpha)$ , ovvero,  $P(Z = 1) = \alpha$ . Immaginiamo  $P(X = 1) = p$

Allora,  $P(Y = 1) = P(X = 1)P(Z = 1) = \alpha p$ , dato che  $1 = 1 \times 1, 0$  altrimenti.

$$\begin{aligned} I(X; Y) &= H(Y) - H(Y|X) \\ &= H(Y) - P(X = 1)H(Z) \\ &= H(\alpha p) - pH(\alpha) \end{aligned}$$

Se volessimo trovare la distribuzione che massimizza l'informazione mutuale:

$$p^* = \frac{1}{\alpha(2^{\frac{H(\alpha)}{\alpha}} + 1)}$$

## Canali simmetrici

Consideriamo un canale con la seguente transition matrix:

$$p(y|x) = \begin{bmatrix} 0.3 & 0.2 & 0.5 \\ 0.5 & 0.3 & 0.2 \\ 0.2 & 0.5 & 0.3 \end{bmatrix}.$$

Qua la entry della  $x$ -esima riga e della  $y$ -esima colonna denota la probabilità condizionale che  $y$  sia ricevuto quando  $x$  è mandato. In questo canale tutte le righe sono permutazioni di altre e così sono anche le colonne. Questo canale è perciò detto **simmetrico**.

Possiamo trovare con facilità un'espressione per la capacità del canale.

Se mettiamo  $r$  la riga della transition matrix, abbiamo:

$$\begin{aligned} I(X; Y) &= H(Y) - H(Y|X) \\ &= H(Y) - H(r) \\ &\leq \log |\Upsilon| - H(r) \end{aligned}$$

Con l'uguaglianza se la distribuzione in output è uniforme.

Dato che  $p(x) = \frac{1}{|\chi|}$  ha una distribuzione uniforme su  $\Upsilon$ , allora la capacità del canale:

$$C = \max_{p(x)} I(X; Y) = \log 3 - H(0.5, 0.3, 0.2)$$

E  $C$  si raggiunge con una distribuzione uniforme sull'input.

## Definizione di canale simmetrico e debolmente simmetrico

Un canale è simmetrico se le righe della matrice di transizione del canale sono permutazioni tra loro, così come le colonne. Un canale è detto *debolmente simmetrico* se ogni riga della matrice di transizione è una permutazione di ogni altra riga mentre la somma delle colonne sono uguali.

*Canale Weakly Symmetric*:

$$p(y|x) = \begin{pmatrix} \frac{1}{3} & \frac{1}{6} & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & \frac{1}{6} \end{pmatrix}$$

## Teorema del canale debolmente simmetrico

Per un canale debolmente simmetrico,

$$C = \log |\Upsilon| - H(\text{row of transition matrix})$$

## Proprietà della channel capacity

- $C \geq 0$  visto che  $I(X; Y) \geq 0$
- $C \leq \log |\chi|$  visto che  $C = \max I(X; Y) \leq \max H(X) = \log |\chi|$
- $C \leq \log |\Upsilon|$
- $I(X; Y)$  è una funzione continua di  $p(x)$
- $I(X; Y)$  è una funzione concava di  $p(x)$

## Introduzione al Channel Coding Theorem

### Canale discreto

Un canale discreto, scritto  $(\chi, p(y|x), \Upsilon)$ , consiste in due alfabeti *i/o* finiti  $\chi$  e  $\Upsilon$  e una collezione di p.m.f.  $p(y|x)$ , una per ogni  $x$  in  $\chi$ , tale che  $\forall x, y \quad p(y|x) \geq 0$  e  $\forall x \quad \sum_y p(y|x) = 1$  dove diamo per scontato che  $X$  sia l'input e  $Y$  sia l'output del canale.

Un canale discreto può avere un'estensione n-esima, qui sotto la definizione originale:

**Definition** The *n*th extension of the discrete memoryless channel (DMC) is the channel  $(\mathcal{X}^n, p(y^n|x^n), \mathcal{Y}^n)$ , where

$$p(y_k|x^k, y^{k-1}) = p(y_k|x_k), \quad k = 1, 2, \dots, n. \quad (7.26)$$

### CANALE DISCRETO SENZA FEEDBACK

Se un canale è senza feedback (ovvero se i simboli in input non dipendono dai simboli in output precedenti), la channel transition function per la *n*-esima estensione del canale discreto memoryless si riduce a:

$$p(y^n|x^n) = \prod_{i=1}^n p(y_i|x_i)$$

E da ora in poi quando ci riferiamo a canali discreti memoryless intenderemo senza feedback.

### Codice per un canale

Un codice  $(M, n)$  per un canale  $(\chi, p(y|x), \Upsilon)$  consiste in:

- Un index set  $\{1, 2, \dots, M\}$
- Una funzione di **encoding**  $X^n : \{1, 2, \dots, M\} \rightarrow \chi^n$ , restituendo codewords del tipo  $x^n(1) \dots x^n(M)$ . Il set di codewords viene detto codebook.
- Una funzione di **decoding**  $g : \Upsilon^n \rightarrow \{1, 2, \dots, M\}$  che è una regola deterministica per assegnare una guess ad ogni possibile vettore ricevuto

### Probabilità condizionale di errore

$$\lambda_i = Pr(g(Y^n) \neq i | X^n = x^n(i)) = \sum_{y^n} p(y^n|x^n(i)) I(g(y^n) \neq i)$$

Dove  $i$  è l'indice e  $I(\cdot)$  è la funzione indicatore.

*Probabilità che sia avvenuto un errore, ovvero, la probabilità che il simbolo di decoding non corrisponda al simbolo che dopo l'encoding il canale ha provveduto a mandare.*

### Probabilità massimale di errore $\lambda^{(n)}$

$$\lambda^{(n)} = \max_{i \in \{1, 2, \dots, M\}} \lambda_i$$

### Probabilità di errore media $P_e^{(n)}$

$$P_e^{(n)} = \frac{1}{M} \sum_{i=1}^M \lambda_i$$

### Rateo $R$ per un codice $(M, n)$

$$R = \frac{\log M}{n} \text{ bits per transmission}$$

## Rateo achievable

Un rate è *achievable* se esiste una sequenza di  $(\lceil 2^{nR} \rceil, n)$  codici tali che la probabilità massimale di errore  $\lambda^{(n)}$  tende a 0 quando n tende a infinito.

## Capacità di un canale

La capacità di un canale è rateo assolutamente più alto.

Ratei minori della capacità restituiscono probabilità di errore arbitrariamente piccole per lunghezze di blocchi grandi.

## Jointly typical sequences

Facciamo decoding di un output di un canale  $Y^n$  come l'i-esimo indice se la codeword  $X^n(i)$  è "jointly typical" con il segnale ricevuto  $Y^n$ . Vediamo ora cosa significa che questo sia j.t..

Un set  $A_\epsilon^{(n)}$  di *jointly typical* sequences  $\{(x^n, y^n)\}$  rispetto alla distribuzione  $p(x, y)$  è il set di n-sequenze con entropie empiriche  $\epsilon$ -close alle entropie vere:

$$A_\epsilon^{(n)} = \left\{ (x^n, y^n) \in \mathcal{X}^n \times \mathcal{Y}^n : \right. \\ \left| -\frac{1}{n} \log p(x^n) - H(X) \right| < \epsilon,$$

$$\left. \begin{aligned} \left| -\frac{1}{n} \log p(y^n) - H(Y) \right| &< \epsilon, \\ \left| -\frac{1}{n} \log p(x^n, y^n) - H(X, Y) \right| &< \epsilon \end{aligned} \right\},$$

where

$$p(x^n, y^n) = \prod_{i=1}^n p(x_i, y_i).$$

## Joint AEP

**Theorem 7.6.1** (Joint AEP) Let  $(X^n, Y^n)$  be sequences of length  $n$  drawn i.i.d. according to  $p(x^n, y^n) = \prod_{i=1}^n p(x_i, y_i)$ . Then:

1.  $\Pr((X^n, Y^n) \in A_\epsilon^{(n)}) \rightarrow 1$  as  $n \rightarrow \infty$ .
2.  $|A_\epsilon^{(n)}| \leq 2^{n(H(X,Y)+\epsilon)}$ .
3. If  $(\tilde{X}^n, \tilde{Y}^n) \sim p(x^n)p(y^n)$  [i.e.,  $\tilde{X}^n$  and  $\tilde{Y}^n$  are independent with the same marginals as  $p(x^n, y^n)$ ], then

$$\Pr\left((\tilde{X}^n, \tilde{Y}^n) \in A_\epsilon^{(n)}\right) \leq 2^{-n(I(X;Y)-3\epsilon)}. \quad (7.39)$$

Also, for sufficiently large  $n$ ,

$$\Pr\left((\tilde{X}^n, \tilde{Y}^n) \in A_\epsilon^{(n)}\right) \geq (1-\epsilon)2^{-n(I(X;Y)+3\epsilon)}. \quad (7.40)$$

Questa sezione è sul libro 1 a pagina 196. Non viene riportata qui perché lunga e laboriosa.

## Channel coding theorem

Per un canale discreto memoryless, tutti i ratei sotto la capacità  $C$  sono *achievable*. Specificatamente, per ogni rateo  $R < C$ , esiste una sequenza di  $(2^{nR}, n)$  codici con massima probabilità di errore  $\lambda^{(n)}$  che tende a zero.

Al contrario, ogni sequenza di  $(2^{nR}, n)$  codici con  $\lambda^{(n)} \rightarrow 0$  deve avere  $R \leq C$ .

*Proof:* dobbiamo provare che ratei  $R < C$  sono *achievable*.

Fissiamo  $p(x)$ . Generiamo un codice randomico secondo la distribuzione  $p(x)$ .

Specificatamente, generiamo  $2^{nR}$  codewords secondo la distribuzione:

$$p(x^n) = \prod_{i=1}^n p(x_i)$$

Mostriamo qui le  $2^{nR}$  codewords come righe di una matrice:

$$\mathcal{C} = \begin{bmatrix} x_1(1) & x_2(1) & \cdots & x_n(1) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(2^{nR}) & x_2(2^{nR}) & \cdots & x_n(2^{nR}) \end{bmatrix}.$$

La probabilità che generiamo un codice particolare  $C$  è

$$Pr(C) = \prod_{w=1}^{2^{nR}} \prod_{i=1}^n p(x_i(w)).$$

Consideriamo le seguenti sequenze di eventi:

1. Un codice  $C$  è generato secondo  $p(x)$
2. Il codice  $C$  è rivelato sia al sender che al receiver. Entrambi conoscono la channel transition matrix  $p(y|x)$  per il canale
3. Il messaggio  $W$  è scelto secondo una distribuzione uniforme

$$Pr(W = w) = 2^{-nR}$$

4. La  $w$ -esima codeword  $X^n(w)$ , che corrisponde alla  $w$ -esima riga di  $C$ , viene inviata sul canale
5. Il ricevente ottiene la sequenza  $Y^n$  secondo la distribuzione

$$P(y^n|x^n(w)) = \prod_{i=1}^n p(y_i|x_i(w)).$$

6. Il ricevente indovina quale messaggio è stato mandato secondo il processo di **jointly typical decoding**: il ricevente dichiara che l'indice  $\hat{W}$  è stato mandato se le condizioni seguenti sono soddisfatte
  1.  $(X^n(\hat{W}), Y^n)$  è jointly typical
  2. Non esiste nessun altro indice  $W' \neq \hat{W}$  tale che  $(X^n(W'), Y^n) \in A_\epsilon^{(n)}$
7. Se queste condizioni **non** sono soddisfatte, abbiamo un errore. Se  $\hat{W} \neq W$  abbiamo un errore. Scriviamo  $\xi$  l'evento tale.

**Analisi della probabilità di errore**: invece di calcolare la probabilità dell'errore per un codice singolo, dobbiamo calcolare la media su tutti i codici generati randomicamente secondo la distribuzione. Per la simmetria della costruzione del codice, la probabilità media di errore non dipende da un indice particolare che è stato inviato.

Per una codeword tipica, ci sono due diverse fonti di errore quando utilizziamo il **jointly typical decoding**: o l'output  $Y^n$  non è jointly typical con la codeword trasmessa oppure c'è un'altra codeword che è jointly typical con  $Y^n$ . Come si vede dalla joint AEP (che non ho scritto sopra), la probabilità che una codeword e la sequenza ricevuta siano j.t. è 1.

Per ogni codeword **rivale**, la probabilità che sia j.t. con la sequenza ricevuta è approssimativamente  $2^{-nI}$ , quindi possiamo usare  $2^{nI}$  codewords ed avere comunque una bassa probabilità di errore.

**Calcolo dettagliato della probabilità di errore**:  $W$  è fatta secondo una distribuzione uniforme su  $\{1, 2, \dots, 2^{nR}\}$  e utilizziamo il j.t. decoding  $\hat{W}(Y^n)$  come nello step 6 precedente. Sia  $\xi = \{\hat{W}(Y^n) \neq W\}$  l'evento che denota l'errore. Calcoliamo la probabilità media d'errore, che è una media su tutte le codewords del codebook su tutti i codebooks, allora:

$$\begin{aligned}
\Pr(\mathcal{E}) &= \sum_{\mathcal{C}} \Pr(\mathcal{C}) P_e^{(n)}(\mathcal{C}) \\
&= \sum_{\mathcal{C}} \Pr(\mathcal{C}) \frac{1}{2^{nR}} \sum_{w=1}^{2^{nR}} \lambda_w(\mathcal{C}) \\
&= \frac{1}{2^{nR}} \sum_{w=1}^{2^{nR}} \sum_{\mathcal{C}} \Pr(\mathcal{C}) \lambda_w(\mathcal{C}),
\end{aligned}$$

Dove  $P_e^{(n)}(\mathcal{C})$  è definita per j.t. decoding. Per la simmetria della costruzione del codice, questo errore non dipende da un indice particolare inviato, quindi possiamo dire che  $W = 1$  è stato inviato, visto che:

$$\begin{aligned}
\Pr(\mathcal{E}) &= \frac{1}{2^{nR}} \sum_{w=1}^{2^{nR}} \sum_{\mathcal{C}} \Pr(\mathcal{C}) \lambda_w(\mathcal{C}) \\
&= \sum_{\mathcal{C}} \Pr(\mathcal{C}) \lambda_1(\mathcal{C}) \\
&= \Pr(\mathcal{E}|W = 1).
\end{aligned}$$

Definiamo quindi i seguenti eventi:

$$E_i = \{(X^n(i), Y^n) \in A_\epsilon^{(n)}\}, \quad i \in \{1, 2, \dots, 2^{nR}\}$$

Dove  $E_i$  è l'evento che la  $i$ -esima codeword e  $Y^n$  siano j.t.. Ricordiamo che  $Y^n$  è il risultato del mandare la prima codeword  $X^n(1)$  sul canale.

Quindi un errore avviene nello schema di decoding o se  $E_1^C$  avviene (la codeword e la sequenza ricevuta non sono j.t.) oppure  $E_2 \cup E_3 \cup \dots \cup E_{2^{nR}}$  avviene (quando una codeword sbagliata è j.t. con la sequenza ricevuta). Se denotiamo con  $P(\xi)$  la probabilità  $Pr(\xi|W = 1)$ , abbiamo:

$$\begin{aligned}
\Pr(\mathcal{E}|W = 1) &= P(E_1^c \cup E_2 \cup E_3 \cup \dots \cup E_{2^{nR}}|W = 1) \\
&\leq P(E_1^c|W = 1) + \sum_{i=2}^{2^{nR}} P(E_i|W = 1),
\end{aligned}$$

Per l'unione degli eventi. Ora, per la joint AEP,  $P(E_1^C|W = 1) \rightarrow 0$ , quindi:

$$P(E_1^c | W = 1) \leq \epsilon \quad \text{per } n \text{ abbastanza grande}$$

E per il processo di generazione di codice  $X^n(1)$  e  $X^n(i)$  sono indipendenti per  $i \neq 1$ , e così anche  $Y^n$  e  $X^n(i)$ . Dunque la probabilità che  $X^n(i)$  e  $Y^n$  siano j.t. è  $\leq 2^{-n(I(X;Y)-3\epsilon)}$  per la jointly AEP. Conseguentemente:

$$\begin{aligned} \Pr(\mathcal{E}) &= \Pr(\mathcal{E} | W = 1) \leq P(E_1^c | W = 1) + \sum_{i=2}^{2^n R} P(E_i | W = 1) \\ &\leq \epsilon + \sum_{i=2}^{2^n R} 2^{-n(I(X;Y)-3\epsilon)} \\ &= \epsilon + (2^{nR} - 1) 2^{-n(I(X;Y)-3\epsilon)} \\ &\leq \epsilon + 2^{3n\epsilon} 2^{-n(I(X;Y)-R)} \\ &\leq 2\epsilon \end{aligned}$$

Per  $n$  sufficientemente grande e  $R < I(X; Y) - 3\epsilon$ . E quindi visto che  $R < I(X; Y)$  possiamo scegliere un  $\epsilon$  adeguato e  $n$  così che la probabilità di errore media sia minore di  $2\epsilon$ .

Per finire la *proof*, andremo a rafforzare questa conclusione con una serie di selezioni di codici:

1. Scegliamo  $p(x)$  nella proof come  $p^*(x)$ , la distribuzione su  $X$  che arriva alla capacità- Allora la condizione  $R < I(X; Y)$  può essere sostituita con  $R < C$ , che è la achievability condition.
2. Rimuoviamo la media su tutti i codebooks. Visto che la probabilità media di errore su un codebook è piccola ( $\leq 2\epsilon$ ), esiste allora almeno 1 codenook  $C^*$  con una probabilità di errore media. E quindi,  $Pr(\xi|C^*) \leq 2\epsilon$ . Per trovare  $C^*$  facciamo una ricerca esaustiva su tutti i codici.
3. Buttiamo via la metà meno importante delle codeword nel miglior codebook  $C^*$ . Visto che la media aritmetica della probabilità di errore  $P_e^{(n)}(C^*)$  per il codice è minore di  $2\epsilon$ , abbiamo  $Pr(\xi|C^*) \leq \frac{1}{2^{nR}} \sum \lambda_i(C^*) \leq 2\epsilon$  che implica che almeno la metà degli indici  $i$  e le codewords associate  $X^n(i)$  devono avere la probabilità condizionata di errore  $\lambda_i$  minore di  $4\epsilon$ . Dunque la metà migliore delle codewords ha una probabilità massimale di errore minore di  $4\epsilon$ .

Combinando questi miglioramenti abbiamo costruito un codice con rateo  $R' = R - \frac{1}{n}$  con una probabilità massimale di errore  $\lambda(n) \leq 4\epsilon$ . Questo prova l'achievability di qualsiasi rateo sotto la capacità.

## Zero error codes

E se fossimo in un contesto dove assumiamo che nessun errore sia permesso?  
Dimostriamo dunque che:

$$P_e^{(n)} \implies R \leq C$$

Assumiamo di avere un codice  $(2^{nR}, n)$  con probabilità di errore 0 (questo vuol dire che l'output del decoder  $g(Y^n)$  è uguale all'input  $W$  con probabilità 1). Allora l'indice input  $W$  è determinato dalla sequenza di output. Ora, per ottenere un legame forte, assumiamo arbitrariamente che  $W$  sia uniformemente distribuita su  $\{1, 2, \dots, 2^{nR}\}$ , e dunque  $H(W) = nR$ . Possiamo dunque scrivere la stringa di diseguaglianza:

$$\begin{aligned} nR &= H(W) = \underbrace{H(W|Y^n)}_{=0} + I(W; Y^n) \\ &= I(W; Y^n) \\ &\stackrel{(a)}{\leq} I(X^n; Y^n) \\ &\stackrel{(b)}{\leq} \sum_{i=1}^n I(X_i; Y_i) \\ &\stackrel{(c)}{\leq} nC, \end{aligned}$$

Dove (a) segue dalla *data-processing inequality*, (b) la proveremo successivamente nel lemma 7.9.2 assumendo che sia discreto memoryless e (c) segue dalla definizione di capacità. Dunque, per ogni zero-error code, per ogni  $n$  vale:

$$R \leq C \quad , \square.$$

## Diseguaglianza di Fano e l'inverso del coding theorem

Estendiamo ora la *proof* derivata dai zero-error codes al caso di codici con probabilità di errore molto piccole. Il nuovo ingrediente sarà la diseguaglianza di Fano, che ci da un lower bound sulla probabilità di errore in termini di entropia condizionata. Qua, un'immagine della *proof* della diseguaglianza di Fano:

Let us define the setup under consideration. The index  $W$  is uniformly distributed on the set  $\mathcal{W} = \{1, 2, \dots, 2^{nR}\}$ , and the sequence  $Y^n$  is related probabilistically to  $W$ . From  $Y^n$ , we estimate the index  $W$  that was sent. Let the estimate be  $\hat{W} = g(Y^n)$ . Thus,  $W \rightarrow X^n(W) \rightarrow Y^n \rightarrow \hat{W}$  forms a Markov chain. Note that the probability of error is

$$\Pr(\hat{W} \neq W) = \frac{1}{2^{nR}} \sum_i \lambda_i = P_e^{(n)}. \quad (7.88)$$

We begin with the following lemma, which has been proved in Section 2.10:

**Lemma 7.9.1** (*Fano's inequality*) *For a discrete memoryless channel with a codebook  $\mathcal{C}$  and the input message  $W$  uniformly distributed over  $2^{nR}$ , we have*

$$H(W|\hat{W}) \leq 1 + P_e^{(n)}nR. \quad (7.89)$$

**Proof:** Since  $W$  is uniformly distributed, we have  $P_e^{(n)} = \Pr(W \neq \hat{W})$ . We apply Fano's inequality (Theorem 2.10.1) for  $W$  in an alphabet of size  $2^{nR}$ .  $\square$

Proveremo ora un lemma che mostra che la capacità di trasmissione non aumenta se utilizziamo un canale discreto *memoryless* più volte:

### Lemma 7.9.2

Sia  $Y^n$  il risultato di passare  $X^n$  attraverso un canale discreto memoryless di capacità  $C$ . Allora:

$$I(X^n; Y^n) \leq nC \quad \text{for all } p(x^n). \quad (7.90)$$

#### Proof

$$I(X^n; Y^n) = H(Y^n) - H(Y^n|X^n) \quad (7.91)$$

$$= H(Y^n) - \sum_{i=1}^n H(Y_i|Y_1, \dots, Y_{i-1}, X^n) \quad (7.92)$$

$$= H(Y^n) - \sum_{i=1}^n H(Y_i|X_i), \quad (7.93)$$

visto che per definizione di canale discreto memoryless,  $Y_i$  dipende solo da  $X_i$  ed è condizionalmente indipendente da ogni altra cosa. Per continuare la serie di diseguaglianze, abbiamo:

$$\begin{aligned}
I(X^n; Y^n) &= H(Y^n) - \sum_{i=1}^n H(Y_i | X_i) \\
&\leq \sum_{i=1}^n H(Y_i) - \sum_{i=1}^n H(Y_i | X_i) \\
&= \sum_{i=1}^n I(X_i; Y_i) \\
&\leq nC,
\end{aligned}$$

Dove la seconda segue direttamente dal fatto che l'entropia di una collezione di variabili è minore della somma delle loro entropie individuali, e l'ultima segue dalla definizione di capacità. Quindi, abbiamo provato che usare il canale molte volte non aumenta la capacità d'informazione in bit per trasmissione.

### Proof dell'inverso del channel coding theorem

Dobbiamo far vedere che ogni sequenza di codici con  $\lambda^{(n)} \rightarrow 0$  deve avere  $R \leq C$ . Se la probabilità massimale di errore tende a zero, la probabilità media di errore per una sequenza di codici va anch'essa a zero. Per una regola fissata di encoding  $X^n(\cdot)$  e una regola fissata di decoding  $\hat{W} = g(Y^n)$ , abbiamo  $W \rightarrow X^n(W) \rightarrow Y^n \rightarrow \hat{W}$ . Per ogni  $n$ , sia  $W$  estratta secondo una distribuzione uniforme. Visto che  $W$  ha una distribuzione uniforme,  $Pr(\hat{W} \neq W) = P_e^{(n)} = \frac{1}{2^{nR}} \sum_i \lambda_i$ . Dunque:

$$\begin{aligned}
nR &\stackrel{(a)}{=} H(W) \\
&\stackrel{(b)}{=} H(W|\hat{W}) + I(W; \hat{W}) \\
&\stackrel{(c)}{\leq} 1 + P_e^{(n)} nR + I(W; \hat{W}) \\
&\stackrel{(d)}{\leq} 1 + P_e^{(n)} nR + I(X^n; Y^n) \\
&\stackrel{(e)}{\leq} 1 + P_e^{(n)} nR + nC,
\end{aligned}$$

Dove:

- (a) segue dall'assunzione che  $W$  sia uniforme
- (b) è un'identità

- (c) è la diseguaglianza di Fano per  $W$  prendendo almeno  $2^{nR}$  valori
- (d) è la data-processing inequality
- (e) viene dal lemma 7.9.2 dimostrato prima

Se dividiamo per  $n$ , otteniamo:

$$R \leq P_e^{(n)} R + \frac{1}{n} + C.$$

Se  $n \rightarrow \infty$ , vediamo che i primi due termini nella parte destra tenderanno a 0, e quindi:

$$R \leq C$$

E dunque possiamo riscrivere questa diseguaglianza come:

$$P_e^{(n)} \geq 1 - \frac{C}{R} - \frac{1}{nR}.$$

E questo mostra che se  $R > C$  la probabilità di errore è lontana da 0 per un  $n$  abbastanza grande, dunque non possiamo arrivare una probabilità di errore arbitrariamente piccola sopra la capacità.

## Recap

Abbiamo dunque provato il teorema ed il suo inverso, che essenzialmente dicono che quando  $R < C$ , è possibile mandare informazioni con una probabilità di errore abbastanza piccola e quando  $R > C$  allora la probabilità di errore è spinta via da 0.

## Eguaglianza nell'inverso del channel coding theorem

---

Interessante è esaminare le conseguenze delle due cose appena viste. Riprendendo i passi nel caso quando  $P_e = 0$  abbiamo:

$$\begin{aligned} nR &= H(W) \\ &= H(W|\hat{W}) + I(W; \hat{W}) \end{aligned}$$

$$\begin{aligned} &= I(W; \hat{W}) \\ \stackrel{(a)}{\leq} & I(X^n(W); Y^n) \\ &= H(Y^n) - H(Y^n|X^n) \\ &= H(Y^n) - \sum_{i=1}^n H(Y_i|X_i) \\ \stackrel{(b)}{\leq} & \sum_{i=1}^n H(Y_i) - \sum_{i=1}^n H(Y_i|X_i) \\ &= \sum_{i=1}^n I(X_i; Y_i) \\ \stackrel{(c)}{\leq} & nC. \end{aligned}$$

Abbiamo l'uguaglianza in:

- (a) se  $I(Y^n; X^n(W)|W) = 0$  e  $I(X^n; Y^n|\hat{W}) = 0$  che è vero se tutte le codewords sono distinte e se  $\hat{W}$  è una statistica sufficiente per il decoding
- (b) se gli  $Y_i$  sono indipendenti
- (c) se la distribuzione di  $X_i$  è  $p^*(x)$ , la distribuzione su  $X$  che arriva alla capacità
- (inverso) solo se tutte le precedenti sono soddisfatte

Questo indica che un codice zero-error capacity-achieving ha codewords distinte e la distribuzione degli  $Y_i$  deve essere i.i.d. con

$$p^*(y) = \sum_x p^*(x)p(y|x),$$

la distribuzione su  $Y$  indotta dalla distribuzione ottima su  $X$ .

La distribuzione a cui ci riferiamo nella *converse* è la distribuzione empirica su  $X$  e  $Y$  indotta da una distribuzione uniforme sulle codewords, che sarebbe:

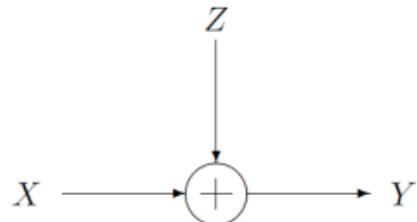
$$p(x_i, y_i) = \frac{1}{2^{nR}} \sum_{w=1}^{2^{nR}} I(X_i(w) = x_i)p(y_i|x_i).$$

Gli esempi di codici con tale capacità raggiunta sono sul libro, pagina 209 e 210.

## Esercizi

### Esercizio 1: capacity

An additive noise channel. Find the channel capacity of the following discrete memoryless channel:



where  $\Pr\{Z = 0\} = \Pr\{Z = a\} = \frac{1}{2}$ . The alphabet for  $x$  is  $\mathbf{X} = \{0, 1\}$ . Assume that  $Z$  is independent of  $X$ .

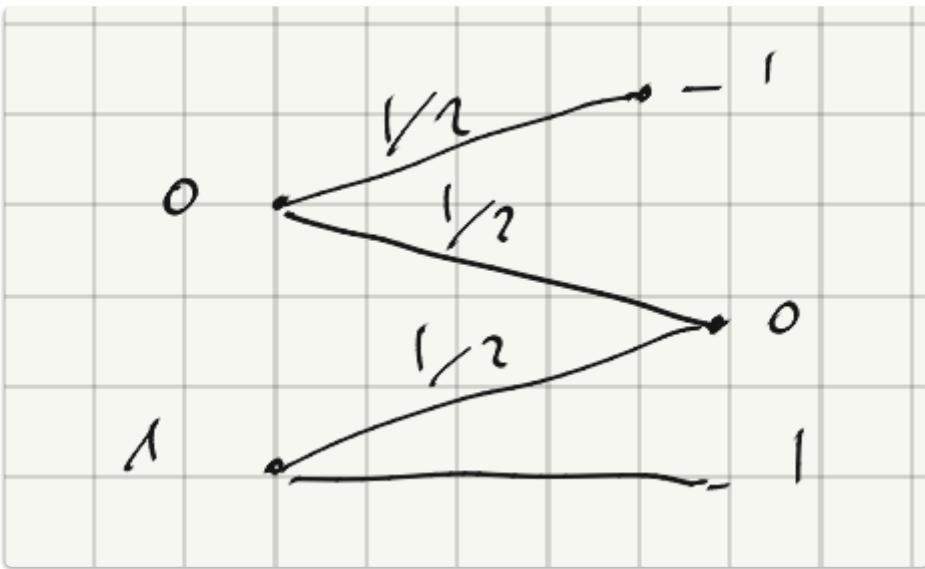
Observe that the channel capacity depends on the value of  $a$ .

Osserviamo che:

- $Y = X + Z$
- $X \in \{0, 1\}$
- $Z \in \{0, a\}$

Distinguiamo ora tutte le possibilità:

- Se  $a = 0$  allora  $Y = X$  dato che  $Z = 0$  quindi per la regola  $C = \max I(X; Y) = I(X; X) = H(X) = 1$
- Se  $a \neq 0, a \neq \pm 1$  allora  $Y$  ha 4 possibili valori:  $0, 1, a, 1 + a$  e dato che conoscendo  $Y$  conosciamo  $X$  vale  $H(X|Y) = 0$ , dunque:  $C = \max I(X; Y) = 1$
- Se  $a = 1$  allora  $Y$  prende 3 possibili valori:  $0, 1, 2$ , dunque ci troviamo nella stessa situazione del canale con cancellazione di bit, dunque  $C = 1 - \alpha = 1/2$
- Se  $a = -1$  allora  $Y$  prende altri 3 possibili valori:  $-1, 0, 1$  e siamo nello stesso identico caso del precedente, come si vede in foto



### Esercizio 3

**Channel capacity.** Consider the discrete memoryless channel  $Y = X + Z \pmod{11}$ , where

$$Z = \begin{pmatrix} 1, & 2, & 3 \\ 1/3, & 1/3, & 1/3 \end{pmatrix}$$

and  $X \in \{0, 1, \dots, 10\}$ . Assume that  $Z$  is independent of  $X$ .

- (a) Find the capacity.
- (b) What is the maximizing  $p^*(x)$ ?

- (a) Assumendo  $Y$  e  $X$  dist. uniformi,  $H(Y|X) = H(Z) = \log 3$  e quindi

$$C = \max_{p(x)} H(Y) - \log 3 \leq \log 11 - \log 3$$

- (b)  $p(X = i) = \frac{1}{11}$ , dato che abbiamo detto uniforme  $X$ .

### Esercizio 4

**Noisy typewriter.** Consider a 26-key typewriter.

- (a) If pushing a key results in printing the associated letter, what is the capacity  $C$  in bits?
- (b) Now suppose that pushing a key results in printing that letter or the next (with equal probability). Thus  $A \rightarrow A$  or  $B, \dots, Z \rightarrow Z$  or  $A$ . What is the capacity?
- (c) What is the highest rate code with block length one that you can find that achieves zero probability of error for the channel in part (b) ?

- (a) Considerando una dist. uniforme della macchina da scrivere, allora  $C = \log 26$
- (b) Come il problema visto,  $C = \log 13$

- (c) Un codice senza errori semplice è un codice che usa lettere in maniera alternata, ovvero  $A, C, E \dots$ , il rateo di questo codice è

$$R = \frac{\log(\#\text{codewords})}{\text{blocklength}} = \frac{\log 13}{1} = \log 13$$

### Esercizio 5

**Zero-error capacity.** A channel with alphabet  $\{0, 1, 2, 3, 4\}$  has transition probabilities of the form

$$p(y|x) = \begin{cases} 1/2 & \text{if } y = x \pm 1 \pmod{5} \\ 0 & \text{otherwise.} \end{cases}$$

- (a) Compute the capacity of this channel in bits.

Il canale è simmetrico, quindi  $H(Y|X) = 1$  e dunque

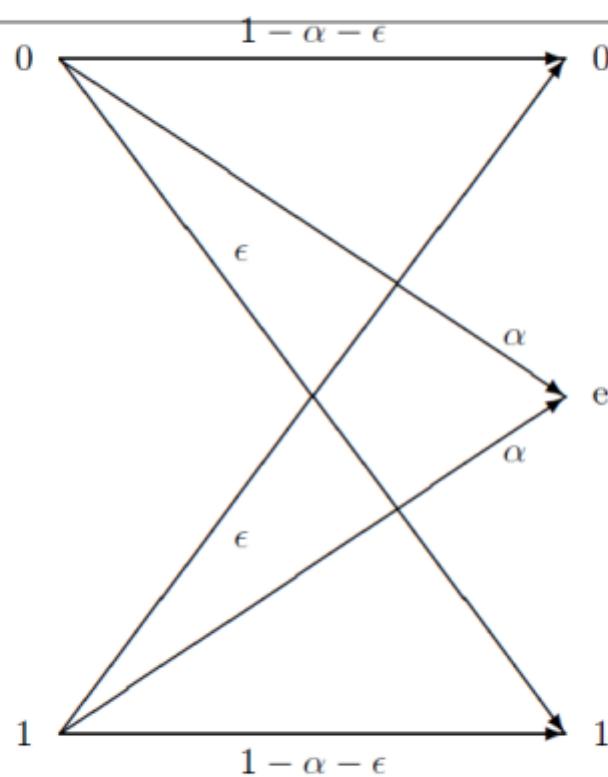
$$I(X; Y) = H(Y) - H(Y|X) = H(Y) - 1$$

La mutua informazione è massimizzata quando  $Y$  è distribuita uniformemente e  $X$  anche, e quindi la capacità in bit è:

$$C = \log_2 5 - 1 = \log_2 2.5 = 1.32$$

### Esercizio 6

**Erasures and errors in a binary channel.** Consider a channel with binary inputs that has both erasures and errors. Let the probability of error be  $\epsilon$  and the probability of erasure be  $\alpha$ , so the channel is as illustrated below:



- (a) Find the capacity of this channel.
- (b) Specialize to the case of the binary symmetric channel ( $\alpha = 0$ ).
- (c) Specialize to the case of the binary erasure channel ( $\epsilon = 0$ ).

- (a)  $C = \max_{p(x)} H(Y) - H(1 - \alpha - \epsilon, \alpha, \epsilon)$
- (b)  $C = 1 - H(\epsilon)$
- (c)  $C = 1 - \alpha$

### Esercizio 7

**Channels with dependence between the letters.** Consider the following channel over a binary alphabet that takes in two bit symbols and produces a two bit output, as determined by the following mapping:  $00 \rightarrow 01$ ,  $01 \rightarrow 10$ ,  $10 \rightarrow 11$ , and  $11 \rightarrow 00$ . Thus if the two bit sequence  $01$  is the input to the channel, the output is  $10$  with probability 1. Let  $X_1, X_2$  denote the two input symbols and  $Y_1, Y_2$  denote the corresponding output symbols.

- (a) Calculate the mutual information  $I(X_1, X_2; Y_1, Y_2)$  as a function of the input distribution on the four possible pairs of inputs.
- (b) Show that the capacity of a pair of transmissions on this channel is 2 bits.
- (c) Show that under the maximizing input distribution,  $I(X_1; Y_1) = 0$ .

Thus the distribution on the input sequences that achieves capacity does not necessarily maximize the mutual information between individual symbols and their corresponding outputs.

- (a) Dato che siamo in un canale *noiseless*,

$$\begin{aligned} I(X_1, X_2; Y_1, Y_2) &= H(Y_1, Y_2) - H(Y_1, Y_2 | X_1, X_2) \\ &= H(Y_1, Y_2) - 0 \\ &= H(p_{00}, p_{01}, p_{10}, p_{11}) \end{aligned}$$

- (b) La massima  $C$  viene raggiunta quando abbiamo dist. uniforme in input che produce sui simboli in output una dist. uniforme
- (c) Su una dist. uniforme, guardiamo la distribuzione congiunta  $X_1 X_2$  e  $Y_1 Y_2$

$X_1 X_2 \backslash Y_1 Y_2$	00	01	10	11
00	0	$\frac{1}{4}$	0	0
01	0	0	$\frac{1}{4}$	0
10	0	0	0	$\frac{1}{4}$
11	$\frac{1}{4}$	0	0	0

Da questa possiamo ricavare la distribuzione congiunta  $X_1$  e  $Y_1$ :

$X_1 \backslash Y_1$	0	1
0	$\frac{1}{4}$	$\frac{1}{4}$
1	$\frac{1}{4}$	$\frac{1}{4}$

Ovvero la dist. congiunta è un prodotto di marginali, ovvero  $X$  e  $Y$  sono indipendenti, quindi la loro mutua informazione è 0.

### Esercizio 8

**Channel capacity:** Calculate the capacity of the following channels with probability transition matrices:

(a)  $\mathcal{X} = \mathcal{Y} = \{0, 1, 2\}$

$$p(y|x) = \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix} \quad (7.87)$$

(b)  $\mathcal{X} = \mathcal{Y} = \{0, 1, 2\}$

$$p(y|x) = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \end{bmatrix} \quad (7.88)$$

(c)  $\mathcal{X} = \mathcal{Y} = \{0, 1, 2, 3\}$

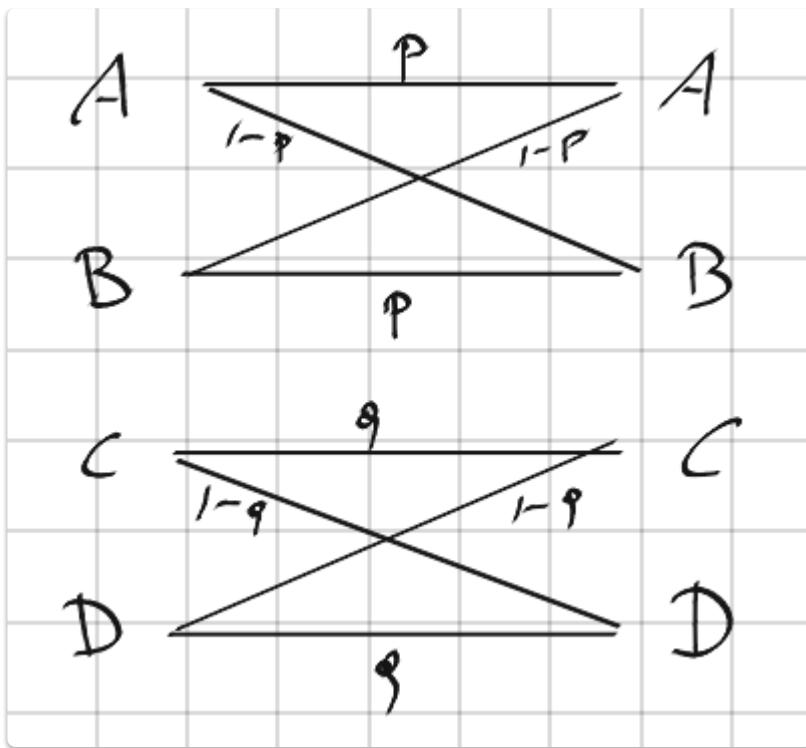
$$p(y|x) = \begin{bmatrix} p & 1-p & 0 & 0 \\ 1-p & p & 0 & 0 \\ 0 & 0 & q & 1-q \\ 0 & 0 & 1-q & q \end{bmatrix} \quad (7.89)$$

**Theorem 7.2.1** *For a weakly symmetric channel,*

$$C = \log |\mathcal{Y}| - H(\text{row of transition matrix}), \quad (7.25)$$

*and this is achieved by a uniform distribution on the input alphabet.*

- (a) Siamo nel caso simmetrico, quindi  $C = \max_{p(x)} I(X; Y) = \log 3 - H(1/3, 1/3, 1/3)$  per il teorema qui sopra.
- (b) Stessa cosa di prima  $C = \log 3 - H(1/2, 1/2, 0)$
- (c) Siamo di fronte a una composizione parallela di due canali simmetrici binari, che da soli avrebbero  $C = 1 - H(p)$ :



Se assumiamo che  $p(A) + p(B) = \alpha$  e  $p(C) + p(D) = 1 - \alpha$  allora possiamo dimostrare che

$$I(X;Y) = H(\alpha) + \alpha I(X;Y|X \in \{A, B\}) + (1 - \alpha)I(X;Y|X \in \{C, D\})$$

La dimostrazione resta sul PDF delle risposte.

# Error Correcting Codes

## Introduzione: il generatore e le parity-check matrices

In generale, il channel coding theorem soffre principalmente di 3 difetti che dobbiamo risolvere riguardo i **codici**:

1. Sono difficili da trovare
2. Sono difficili da analizzare
3. Sono difficili da implementare

Difatti, virtualmente il solo coding scheme tra quelli visti che non soffre di questi difetti è l'Hamming code scheme, e che potremmo utilizzare in questa sezione.

Per semplificare la notazione, dobbiamo strutturare l'alfabeto in input per il canale  $A_x$ : da ora in poi assumiamo che  $A_x$  sia un field finito  $F_q$  con  $q$  elementi, e che una codeword  $x$  di lunghezza  $n$  è un vettore n-dimensionale su  $F_q$ .

### Definizione di codice lineare

Un codice lineare  $(n, k)$  (*colonne,righe*) su  $F_q$  è un sotto-spazio  $k$ -dimensionale di uno spazio vettoriale  $n$ -dimensionale  $V_n(F_q)$ ;  $n$  è la lunghezza del codice,  $k$  la dimensione. Il **rateo** del codice è  $k/n$  ovvero righe/colonne.

Un vantaggio immediato dei codici lineari su quelli non-lineari è che sono molto più semplici da specificare. Un codice lineare  $C$  può essere completamente descritto con ogni set di  $k$  codewords linearmente indipendenti. Se organizziamo le codewords in una matrice  $G$  grande  $k \times n$ , chiamiamo  $G$  una matrice generatrice per  $C$ .

### Definizione di generator matrix

Sia  $C$  un codice lineare  $(n, k)$  su  $F_q$ . Una matrice  $G$  avente spazio di righe eguale a  $C$  è detta una generator matrix per  $C$ . All'inverso, se  $G$  è una matrice con gli elementi da  $F_q$ , il suo spazio di righe è chiamato il codice generato da  $G$ .

### Ottenere codici data una generator matrix

Data una matrice generatrice  $G$  possiamo ottenere le codeword  $w$  di un codice  $C$  come

$$w = sG$$

Quindi data una matrice  $G$  seguente:

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Possiamo generare 4 codewords, dato che saranno  $2^k$  possibili messaggi.

Se i messaggi sono:

- 00: 000000
- 01: 011111
- 10: 100101
- 11: 111010

(NON HO VERIFICATO I CALCOLI, NON SO COME MAI IL PROF LI RIPORTA DIVERSI!)

Specificando una matrice generatrice è possibile dare descrizioni compatte di codici interessanti, vediamo degli esempi:

**Example 7.1** A  $(5, 1)$  linear code  $C_1$  with generator matrix

$$G_1 = [1 \ 1 \ 1 \ 1 \ 1].$$

**Example 7.2** A  $(5, 3)$  linear code  $C_2$  with generator matrix

$$G_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

**Example 7.3** A  $(7, 4)$  linear code  $C_3$  with generator matrix

$$G_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Il secondo vantaggio è la facilità della fase di **encoding**. Un codice lineare  $(n, k)$  ha  $q^k$  codewords, quindi nel caso binario  $2^k$ , e queste possono essere utilizzate per comunicare  $q^k$  possibili messaggi distinti; se diamo per buono che questi messaggi siano indexati da  $q^k$  k-tuple  $u$  e che le righe di  $G$  siano linearmente indipendenti, allora la regola di encoding sarà semplicissima:

$$u \rightarrow uG,$$

Dove  $uG$  denota la moltiplicazione del vettore  $u$  grande  $1 \times k$  per la matrice  $G$  grande  $k \times n$ .

Prendiamo gli esempi sopra e facciamo  $G_2$  per  $C_2$ : il mapping diventerà:

$$(u_1, u_2, u_3) \rightarrow (u_1 + u_3, u_1 + u_3, u_1 + u_2 + u_3, u_2 + u_3, u_3).$$

Beh ora sappiamo anche un'altra cosa interessante: *ogni codice lineare ha una matrice di generazione RRE unica* (row reduced echelon), che ha le seguenti proprietà:

1. L'elemento più a sinistra di ogni riga che non è 0 è 1.
2. Ogni colonna che contiene questo 1 ha tutte le altre entry a 0.
3. Se l'elemento non-zero più a sinistra  $i$  occorre nella colonna  $t_i$ , allora  $t_i < t_2 \dots < t_r$ .

E se guardiamo l'esempio notiamo che  $G_1, G_3$  sono in forma **RRE** mentre  $G_2$  deve essere trasformata in:

$$G'_2 = \begin{bmatrix} \textcircled{1} & 1 & 0 & 0 & 1 \\ 0 & 0 & \textcircled{1} & 0 & 1 \\ 0 & 0 & 0 & \textcircled{1} & 1 \end{bmatrix},$$

Ed il cui encoding diventa dunque:

$$(u_1, u_2, u_3) \rightarrow (u_1, u_1, u_2, u_3, u_1 + u_2 + u_3).$$

Questo encoding ha, tra l'altro, una feature desiderabile: i simboli  $u_1, u_2, u_3$  appaiono chiari nella codeword: in generale, il simbolo  $u_i$  appare come  $t_i$ -esimo componente della codeword  $x = uG$  se l'elemento più a sinistra della riga  $i$ -esima di  $G$  appare nella colonna  $t_i$ .

Ogni codice, lineare o meno, che ha la proprietà appena spiegata è detto **sistematico**, e quindi abbiamo appena dimostrato come **ogni codice lineare è sistematico**.

## Parity-check matrix di $C$

Esiste un'altra matrice interessante che andiamo ad affrontare ora, la **parity check matrix**.

Un **parity check** per un codice  $C$  è un'equazione della forma:

$$a_1x_1 + a_2x_2 + \dots + a_nx_n = 0$$

che è soddisfatto per tutti gli  $x \in C$ . Il set di tutti i vettori  $a$  per i quali è soddisfatto è in-se un sottospazio di  $V_n(F_q)$ . Questo è denotato da  $C^\perp$  ed è detto un **codice duale** di  $C$ . Una parity-check matrix per  $C$  è semplicemente una generator matrix per  $C^\perp$

*Sia  $C$  un codice lineare  $(n, k)$  su  $F_q$ . Una matrice  $H$  con la proprietà  $Hx^T = 0$  iff  $x \in C$  è detta una parity-check matrix per  $C$ .*

Ovviamente, grazie a quello che abbiamo appena detto sappiamo che ogni codice lineare avrà la sua parity-check matrix RRE. Ma è molto più comodo usare una forma canonica di  $H$ . Per esempio se  $G = [I_k A]$  segue che  $H$  può essere creata nel seguente modo:

$$H = [-A^T | I_{n-k}].$$

Se  $G$  non è di questa forma,  $H$  può essere ottenuta facendo una permutazione di colonna per mettere  $G$  nella forma  $[I_k | A]$  e poi facendo la permutazione inversa su  $[-A^T | I_{n-k}]$ .

L'output della parity-check matrix per il codice errato restituisce la **sindrome**, che è possibile checkare con la parity-check matrix per sapere dove è avvenuto l'errore

$$H(\vec{c}^T + \underline{\vec{e}}^T) = \vec{0}$$

*Syndrome*

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Costruiamo le parity-check matrix per gli esempi prima:

$$H_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad H_2 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \end{bmatrix},$$

$$H_3 = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Possiamo dunque scrivere tutto quello che abbiamo detto in un teorema:

### Teorema 7.1

Sia  $C$  un codice lineare  $(n, k)$  su  $F_q$ . Allora esiste una matrice RRE unica  $k \times n$   $H$  con la proprietà che  $x \in C$  sse è nel rowspace di  $G$ . Inoltre, esiste una matrice  $H$  grande  $(n - k) \times n$  con la proprietà che  $x \in C$  sse  $Hx^t = 0$ . Se  $C$  è usata in un canale

*memoryless*, non può esserci una perdita di informazione assumendo che una matrice tale esista e che

$$G = [I_k A], \quad H = [-A^T I_{n-k}],$$

nel qual caso l'encoding di un vettore  $u \in V_k(F_q)$  è dato da  $u \rightarrow (u, uA)$ .

## Syndrome decoding di canali simmetrici q-ari

Fino ad ora non abbiamo incluso nella nostra discussione l'alfabeto dell'output del canale  $A_Y$ . Da ora in poi assumiamo che  $A_Y = F_q$ , ovvero che gli alfabeti in input ed output siano identici, ovvero che i vettori inviati  $x$  e ricevuti  $y$  siano nello stesso spazio  $V_n(F_q)$  e che la differenza  $z = y - x$  sia chiamata **error pattern**. Se  $z_i \neq 0$  allora un errore è occorso nella  $i$ -esima coordinata.

La parity-check matrix  $H$  ci serve per fare decoding in questa situazione. Se  $x$  è trasmesso,  $x$  è una codeword, e così anche  $Hx^T = 0$ . Se il canale ha avuto errori, ovvero,  $z \neq 0$ , allora è altamente probabile che  $Hx^T \neq 0$ . Il vettore  $s = Hy^T$  è chiamato la **syndrome**. La **syndrome** dipende solo dall'error pattern  $\mathbf{z}$  e non dalla codeword trasmessa:

$$\begin{aligned} \mathbf{s} &= H\mathbf{y}^T \\ &= H(\mathbf{x} + \mathbf{z})^T \\ &= H\mathbf{x}^T + H\mathbf{z}^T \\ &= H\mathbf{z}^T, \end{aligned}$$

Dato che  $Hx^T = 0$ . Ovviamente il ricevente non è direttamente interessato a  $z$ , lui vorrebbe sapere  $x$ , ma visto che conosce  $y$  e che  $x = y - z$ , può tranquillamente interessarsi di  $z$ .

La syndrome ci da qualche informazione su  $z$ , ma non abbastanza. Questo perché per un  $s$  fissato, il set di soluzione  $Hz^T = s$  forma un **coset** del codice  $C$ , che sarebbe, un subset di  $V_n(F_q)$  della forma

$$C + z_0 = \{x + z_0 : x \in C\}.$$

Esistono  $q^{n-k}$  cosets di  $C$ , che corrispondono allo stesso numero di syndromes  $s$ . Ogni coset contiene esattamente  $q^k$  elementi. Dunque ogni volta che il ricevente computa  $s$ , ha ridotto la sua ricerca di  $z$  da  $q^n$  a  $q^k$  possibilità, ovvero, il numero di elementi del coset a cui corrisponde  $s$ .

Per far si che si possa distinguere  $z$  tra tutti questi  $q^k$  candidati, è necessario sapere

che il canale è un canale simmetrico q-ario (**qSC**), ovvero, che se  $X$  è un vettore che modella l'input e  $Y$  è il vettore dell'output, allora  $Y = X + Z$ , dove  $Z$  è un vettore i quali componenti sono i.i.d. con una distribuzione comune:

$$P\{Z = 0\} = 1 - (q - 1)\epsilon,$$

$$P\{Z = z\} = \epsilon \quad \text{if } z \neq 0.$$

Per questo canale è facile distinguere i pattern degli errori:

$$Pr(Z = z) = [1 - (q - 1)\epsilon]^{n-w_{H^z}} \epsilon^{w_{H^z}},$$

Dove  $w_{H^z}$  è il **peso Hamming** di  $z$ , definito come il numero di 1 nel codice  $z$ , ovvero il numero di errori in  $z$ !

Se  $\epsilon \leq 1/q$ , la parte destra della formula è una funzione decrescente di  $w_{H^z}$  e quindi lo  $z$  più probabile è quello con il peso minore.

### Algoritmo di decoding syndrome per un qSC

- 
1. Compute the syndrome  $s = Hy^T$ .
  2. Find a minimum-weight vector in the coset corresponding to  $s$ . Call it  $z_0$ .
  3. Output the codeword  $\hat{x} = y - z_0$ .
- 

**Figure 7.1** The syndrome decoding algorithm for a qSC.

Lo step 2 ovviamente rappresenta la parte più importante del lavoro. Nonostante tutto, se  $k$  e  $n - k$  sono piccoli, è possibile implementare lo step 2 come un **table lookup**, che ora andiamo a spiegare.

#### TABLE LOOKUP

Andiamo a prendere il codice  $C_2$  di nuovo. La sua parity-check matrix è la seguente:

$$H = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

In questo caso le possibili syndrome sono: 00, 01, 10, 11. Possiamo classificare i 32 vettori  $z = (z_1, z_2, z_3, z_4, z_5)$  secondo le loro syndrome e organizzarli come mostriamo qui in una matrice  $4 \times 8$ , chiamato lo standard array:

Syndrome	Coset leader								
00	00000	00011	00101	00110	11001	11010	11100	11111	
01	00100	00111	00001	00010	11101	11110	11000	11011	
10	01000	01011	01101	01110	10001	10010	10100	10111	
11	10000	10011	10101	10110	01001	01010	01100	01111	

**Figure 7.2** The standard array for  $C_2$ .

Le righe dello standard array sono i *coset* di  $C$ ; per esempio, la prima riga è il codice stesso! Tra ogni coset un vettore di peso minimo è stato fissato come primo e chiamato *coset leader*. In generale, un elemento in un coset (escludendo il codice stesso) è uguale al coset leader + la codeword che sta sopra.

Ad esempio la entry 01101 nella terza riga è uguale al coset leader 01000 + 00101.

Dato lo standard array, lo step 2 è facile facile: prima della trasmissione formiamo una tabella che consiste di tutte le coppie  $(s, z(s))$ , dove  $s$  è una delle possibili  $q^{n-k}$  syndrome e  $z(s)$  è il leader del coset con syndrome  $s$ . Allora lo step 2 diventa:

$$\text{Metti } z_0 = z(s)$$

Questo algoritmo, se fattibile, è il più veloce conosciuto.

### Come ottenere lo standard array dai codici/generator matrix?

1. Dati i codici  $C$ , che potrebbero essere per esempio  $C = \{00000, 01011, 10100, 11111\}$ , mettiamole nella prima riga dello standard array. Il primo valore 00000, sara' il coset leader mentre associamo, alla sua destra (o sx) la syndrome 000. Ora ci troviamo con la tabella seguente:

Sindrome	Coset Leader			
000	00000	01011	10100	11111

2. Scegliamo una *keyword* con distanza minima. Se 00000 ha distanza 0, troviamo la prima keyword conveniente con distanza 1. Basta dunque che cambio un solo bit, no? Cambiamo quindi il primo bit, e abbiamo il nuovo coset leader 10000.
3. Sommiamo il nuovo coset leader al nostro codice (che si trova in prima fila!):
  1.  $01011 + 10000 = 11011$
  2.  $10100 + 10000 = 00100$
  3.  $11111 + 10000 = 01111$

4. La sindrome della riga si ottiene prendendo poi il nostro  $10000$ , lo trasponiamo, lo moltiplichiamo con  $H$ , la parity-check matrix, ed esce la syndrome, che andiamo ad attaccare di fianco!

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

Sindrome	CoSet Leader			
000	00000	01011	10100	11111
100	10000	11011	00100	01111

5. Da ora cerchiamo un altro vettore con distanza minima (numero minimo di bit cambiati) tale che non e' stato ancora inserito in **nessun'altra casella**! E lo mettiamo nel coset leader: in questo caso 01000 potrebbe essere un'ottima scelta. Facciamo quindi la stessa identica cosa sopra.
6. Ripetiamo questo algoritmo fino al completamento dello standard array!

Sindrome	CoSet leader			
000	00000	01011	10100	11111
100	10000	11011	00100	01111
011	01000	00011	11100	10111
010	00010	01001	10110	11101
001	00001	01010	10101	11110
111	11000	10011	01100	00111
110	10010	11001	00110	01101
101	10001	11010	00101	01110

## Syndrome decoding for dummies

Prendendo spunto da [questo](#) video, ecco una spiegazione semplice di come arrivare alla tabella standard data la generator matrix/parity-check matrix.

La syndrome decoding è utilizzata per correggere errori negli error blocks dati i codici.  
Partiamo dalle definizioni:

- $x$  è il codice mandato,  $y$  è il codice ricevuto
- $z = y - x$  è l'error pattern
- $s = Hy^T$  è la sindrome
- $Hx^T = s$  è il coset del codice  $C$

Partiamo da un esempio, se il codice  $x$  è 10110 viene mandato ma dato il rumore sul canale il codice arriva come  $y = 10100$ , vogliamo sapere quale bit è cambiato da riceventi.

Se sappiamo il nostro error pattern  $z = 00010$  allora basta sommare  $z + y$  per ottenere  $x$

Quindi vorremmo sapere come si trova  $z$ , così da poterlo sommare semplicemente, no?  
Vediamolo da un esempio. Data la parity-check matrix  $H$  qui, calcoliamo tutti i vettori syndrome per tutti i single bit error patterns:

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \quad \begin{array}{l} n=7 \\ k=4 \end{array}$$

Pattern di errori:

1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	1	0
0	0	0	0	0	0	1
0	0	0	0	0	0	1

Ora prendiamo ognuno di queste righe e moltiplichiamolo per  $H$ .

Nel primo caso  $[1000000] \times H^T = 101$ , etc...

I syndrome saranno dunque: 101, 111, 110, 011, 100, 010, 001.

$$S = [1011110] \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= 1+0+1+0+1+0+0 \quad 0+0+1+1+0+1+0 \quad 1+0+0+1+0+0+0$$

$$= 1$$

$$= [110]$$

0

1

110 è la syndrome che identifica la terza riga di  $H^T$ , quindi ora sappiamo che il codice ha avuto un problema sul terzo bit, quindi per correggerlo facciamo

$$1011110 + 0010000 = 1001110 = C$$

## Distanza di Hamming

Uno spazio vettoriale  $V_n(F_q)$  è uno spazio metrico se definiamo la *distanza di Hamming* tra due vettori  $x$  e  $y$  come

$$\begin{aligned} d_H(x, y) &= \text{il numero di componenti per i quali } x_i \neq y_i \\ &= w_H(y - x) \end{aligned}$$

La **distanza di Hamming** tra due stringhe di ugual lunghezza è il numero di posizioni nelle quali i simboli corrispondenti sono diversi. In altri termini, la distanza di Hamming misura il numero di *sostituzioni* necessarie per convertire una stringa nell'altra, o, vista in altro modo, il numero minimo di *errori* che possono aver portato alla trasformazione di una stringa nell'altra.

### Esempio semplice

- La distanza di Hamming tra **101101** e **1001001** è 2.
- La distanza di Hamming tra **2143896** e **2233796** è 3.

Il **peso di Hamming** di una stringa di lunghezza  $k$  è la sua distanza di Hamming dalla stringa costituita da  $k$  zeri. Quindi è il numero di elementi diversi da zero di una stringa: per una stringa binaria è semplicemente il numero di 1; per esempio, il peso di Hamming di 11101 è 4.

Ora andremo ad investigare la relazione tra la geometria di Hamming di un codice e la sua abilità di correggere gli errori su un qSC.

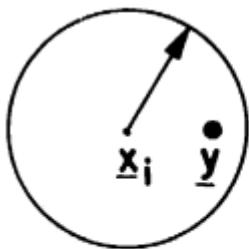
### Correggere gli errori: le sfere di Hamming?

Sia  $C$  un codice di lunghezza  $n$ , non necessariamente lineare, per l'utilizzo su un qSC. Supponiamo di voler  $C$  capace di correggere tutti gli errori di peso Hamming  $\leq e$ ; ovvero, che se mandiamo  $x_i$ , allora  $y = x_i + z$  è ricevuto e  $w_H(z) \leq e$ , vogliamo il nostro output esattamente  $\hat{x} = x_i$ .

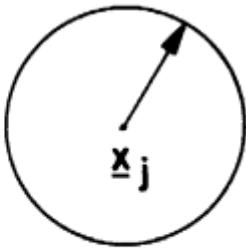
Se ogni codeword è mandata con probabilità  $1/M$  allora la strategia migliore per il ricevente per indovinare cosa il sender ha mandato è quella di prendere quella più vicina ad  $y$ , ovvero, quella per il quale la distanza di Hamming da  $x_i$  è minore.

Ovviamente, se usiamo la geometric decoding strategy, riusciremo a correggere tutti i pattern di peso  $\leq e$  se e solo se la distanza tra ogni coppia di codewords è  $\geq 2e + 1$ .

Se si guarda la figura,



(a)



(b)

**Figure 7.3** Hamming spheres of radius  $e$  around adjacent codewords.

se  $d_H(x_i, x_j) \geq 2e + 1$ , ovvero se le sfere di Hamming di radius  $e$  intorno a  $x_i$  e  $x_j$  sono disgiunte, allora se  $x_i$  viene inviata e  $d_h(x_i, y) \leq e$ ,  $y$  non può essere vicina a  $x_j$  di quanto lo è a  $x_i$ , e quindi il decoder geometrico non preferirà  $x_j$  a  $x_i$ .

Al contrario, se questa distanza è minore di  $2e$ , ovvero se le sfere di Hamming si intersecano, allora è chiaro che se  $x_i$  dovesse essere inviato, esiste un  $y$  la cui distanza di Hamming da  $x_i$  è minore o uguale ad  $e$ .

### Hamming codes: come correggono gli errori (TOTALMENTE OPZIONALE!)

Partiamo dal principio. Nel set di tutti i possibili messaggi che un canale può portare, solo alcuni (un sub-set) sono validi. Quando avviene un errore, allora siamo incaricati di correggere un errore andando a vedere la sua distanza dal suo codice "valido", come faremmo per un tipo (*Hello Wordd* → *Hello World*).

Guardiamo ora esattamente come Hamming avrebbe corretto un blocco da 16 bit, con 4 bit che useremo per "ridondanza":

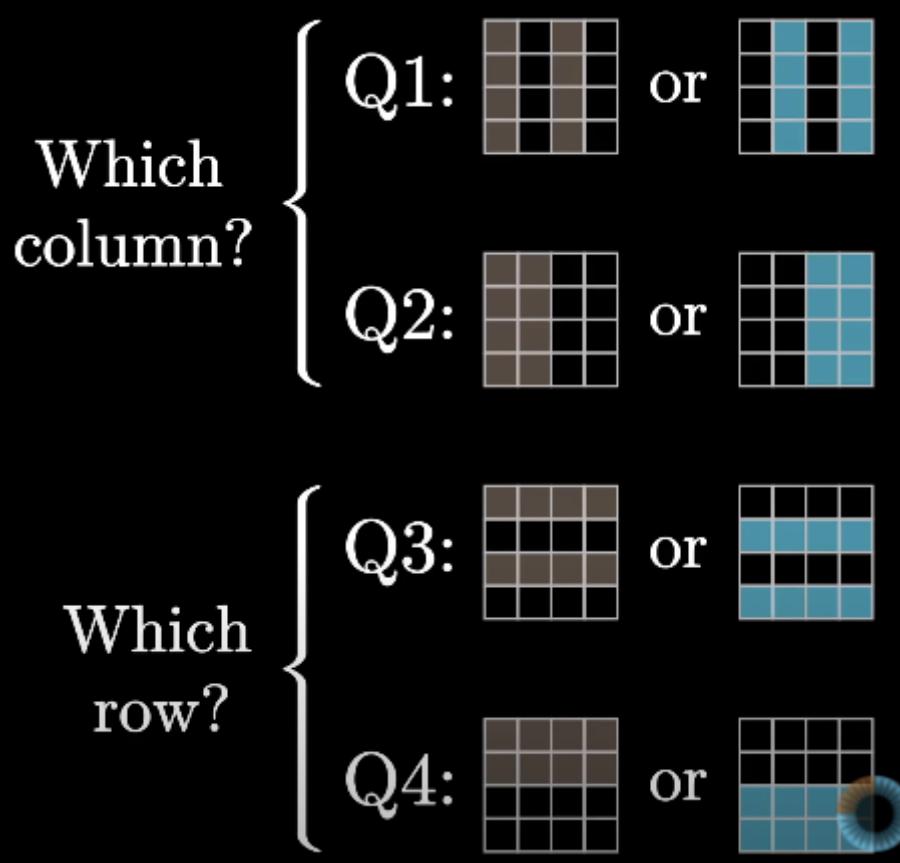
0 0			1 3
	0 5	1 6	0 7
	0 9	1 10	0 11
1 12	0 13	0 14	1 15

Iniziamo dalle basi: **parity-check** bit, cosa sono?

Il primo bit di questa serie ci dirà semplicemente se questo messaggio compone un numero pari oppure dispari, ovvero ci basta contare il numero di "1" nel codice, e se questo è pari oppure dispari mettiamo rispettivamente 0 oppure 1. Il ricevente del nostro messaggio saprà sicuramente che c'è stato un errore (o più di uno!) se questo bit è 0 ma il messaggio contiene un numero dispari di bit o viceversa!

Contemporaneamente, se c'è stato un numero pari di errori, il ricevente non riuscirà solo con questo metodo a capire se c'è stato un errore.

L'idea di Hamming è di limitare questo parity check a dei sub-set di bit nel messaggio, invece di controllarlo completamente: potremmo voler scegliere 1 bit per un set di colonne, 1 bit per un set di righe, un bit per la metà sinistra (o destra) ed un bit per la parte superiore (o inferiore).



Se avviene un errore nella cella 3, allora sarà il parity check Q1 e Q2 che individueranno l'errore, ma non Q3 e Q4.

1 0	1 1	0 2	0 3
0 4	1 5	0 6	0 7
1 8	1 9	0 10	1 11
1 12	0 13	1 14	1 15

Intuitivamente, anche senza aver mai visto di cosa si tratta, possiamo tranquillamente dire che per un codice grande  $n$  abbiamo bisogno di  $\log_2 n$  bits di parità per correggere l'errore per bene. Sul nostro codice  $n = 16$ ,  $\log_2 16 = 4$ , efficiente, vero?

Nel corso, chiameremo questa matrice un (15, 11) Hamming code, dato che il blocco è grande 15 bit, con 4 bit di ridondanza possiamo scrivere 11 bit di messaggio.

Il primo bit, non contato, sarà usato come bit di parità **per l'intero codice**, così da poter avere un 2-error-correcting code, invece che un single-error correcting code.

Facciamo una prova: dato il codice 00110001110 scriviamo il nostro Hamming Code! Sapendo che 1 e tutte le potenze di due sono riservate per la correzione del codice, mettiamo il nostro messaggio nella griglia:

	0	1	2	0 3
4	0 5	1 6	1 7	
8	0 9	0 10	0 11	
1 12	1 13	1 14	0 15	

E ora individuiamo i gruppi e scriviamo i parity bit:

- **G1**: scriviamo 0 (nella posizione 1), questo è il gruppo delle colonne 2 e 4
- **G2**: scriviamo 1 (nella posizione 2), questo è il gruppo delle colonne 3 e 4
- **G3**: scriviamo 1 (nella posizione 4), questo è il gruppo delle righe 2 e 4
- **G4**: scriviamo 1 (nella posizione 8), questo è il gruppo delle righe 3 e 4
- **TOP**: scriviamo 0 (nella posizione 0), questo è il parity bit di tutta la griglia

0 0	0 1	1 2	0 3
1 4	0 5	1 6	1 7
1 8	0 9	0 10	0 11
1 12	1 13	1 14	0 15

Immaginiamo ora di essere il ricevente e ricevere la seguente griglia:

0 0	0 1	1 2	0 3
1 4	0 5	1 6	1 7
1 8	0 9	1 10	0 11
1 12	1 13	1 14	0 15

Controlliamo dunque, uno ad uno, tutti i gruppi:

- Il bit **TOP**, ci dice che la griglia dovrebbe essere pari, ma abbiamo 9 bit ad "1", c'è quindi un errore sicuramente
- G1 ci dice che le colonne 2 e 4 dovrebbero essere pari, ed infatti lo sono. Il nostro errore non è in quelle due colonne
- G2 ci dice che abbiamo avuto un errore, dato che riporta dispari ma abbiamo un numero pari di bit, quindi sappiamo che l'errore è avvenuto nella colonna 3 sicuramente
- G3, ci dice che abbiamo un numero dispari di bit, che è vero, ma ci lascia con i buchi 2 e 10 come "possibili incorretti"

- G4 ci conferma che il bit errato è nella parte bassa della griglia, ovvero nel buco 10, individuando quell'1 che dovrebbe invece essere 0.

0 0 1 0	0 1 0 1	1 1 1 0	0 1 0 1
1 4	0 5	1 6	1 7
1 8	0 9	1 10	0 11
1 12	1 13	1 14	0 15

## Algoritmo di Hamming

1. Numeriamo i bit partendo da 1: 1,2,3,4...
2. Scriviamo questi bit in binario: 1,10,11,100,101,110...
3. Tutte le posizioni che sono potenze di 2, saranno parity bit: 1,2,4,8...
4. Tutte le altre posizioni sono bit che saranno "data bits"
5. Ogni data bit è incluso in un set unico di 2 o più bit di parità:
  1. Il bit di parità 1 copre tutte le posizioni che hanno **l'ultimo bit a 1**: 1,3,5,7,9...
  2. Il bit di parità 2 copre tutte le posizioni che hanno il **penultimo bit a 1**: 2-3,6-7,10-11...
  3. Il bit di parità 4 copre tutte le posizioni che hanno il **terzultimo bit a 1**: 4-7,12-15,20-23...
  4. Il bit di parità 8 copre tutte le posizioni che hanno il **quartultimo bit a 1**: 8-15,24-31, 40-47...

## Distanza minima di un codice

Quindi definiamo la **distanza minima** del codice  $C$  come:

$$d_{\min}(C) = \min\{d_H(x, x') : x, x' \in C, x \neq x'\},$$

Distanza minima: il + piccolo numero di colonne di  $H$  linearmente dipendenti, il numero minimo di colonne che sommate (in binario) fa 0.

E questo è anche la prova del teorema a seguire

## Teorema sulla capacità di correzione del codice

Un codice  $C$  è capace di correggere tutti gli error pattern di peso  $\leq e$  se e solo se  $d_{\min}(C) \geq 2e + 1$ .

Ad esempio un codice con  $d_{\min} = 7$  può correggere tutti gli errori di peso  $\leq 3$ .

Applichiamo dunque questi risultati, ai codici lineari.

Dalla nostra osservazione sappiamo che  $d_H(x, x') = w_H(x - x')$  e  $x - x'$  deve essere una codeword senza zeri se  $C$  è lineare, la distanza minima di un linear code è la stessa del suo peso minimo, dove

$$w_{\min}(C) = \min\{w_H(x) : x \in C, x \neq 0\}$$

E quindi per computare  $d_{\min}$  per un codice lineare  $(n, k)$  non è necessario computare le  $(q^{2k} - q^k)/2$  distanze per  $x \neq x'$ ; sarà sufficiente computare dunque i  $q^k - 1$  pesi  $w_H(x)$  per  $x \neq 0$ .

### Teorema (7.3) della procedura semplice per computare $d_{\min}$

Se  $C$  è un codice lineare  $(n, k)$  su  $F_q$  con una parity-check matrix  $H$ ,  $d_{\min}(C)$  = il numero più piccolo di colonne di  $H$  che sono linearmente dipendenti. Dunque, se ogni subset di  $2t$  o meno colonne di  $H$  è linearmente indipendente, il codice è capace di correggere tutti gli errori di peso  $\leq t$ .

*Nota:* se  $q = 2$  allora le parole "sono linearmente dipendenti" possono essere sostituite come "sommano a 0".

*Proof:* le codewords di  $C$  sono quei vettori  $x$  tali che  $Hx^T = 0$ . Il prodotto  $Hx^T$  è una combinazione lineare delle colonne di  $H$ ; infatti, se  $c_{\dots n}$  sono le colonne di  $H$ ,  $Hx^T = x_1c_1 + \dots + x_nc_n$ . Allora una codeword non-zero di peso  $w$  restituisce una dipendenza non-triviale lineare su  $w$  colonne di  $H$ , e il contrario. Questo prova il teorema,  $\square$ .

## Hamming Codes

Guardiamo la parity-check matrix per  $C_3$ :

$$H_3 = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Applicheremo ora il teorema 7.3 per determinare  $d_{\min}$ .

Sappiamo che quello è il numero più piccolo di colonne dove  $H$  somma a 0.

Chiaramente  $d_{\min} \neq 1, 2$ , visto che le colonne di  $H_3$  non sono a 0 e distinte. Nonostante questo, ci sono molti subset di tre colonne di  $H_3$  che sommano a 0, per esempio, le prime 3 colonne! Quindi,  $d_{\min} = 3$  e così  $C_3$  è un single-error-correcting code, ovvero, è capace di correggere tutti gli errori di peso 0 oppure 1.

Infine, osserviamo che se  $C$  è uno degli  $(n, n - 3)$  single-error-correcting code, allora  $n \leq 7$  visto che se la matrice parity-check grande  $3 \times n$  avesse  $n \geq 8$ , o avrebbe una colonna a 0 (quindi  $d_{\min} = 1$ ) oppure una coppia di colonne identiche (e quindi  $d_{\min} = 2$ ). Mostriamo ora la definizione di codice di Hamming.

## Definizione di Hamming Code

Sia  $H$  una matrice binaria grande  $m \times (2^m - 1)$  tale che le sue colonne sono i  $2^m - 1$  vettori nonzero da  $V_m(F_2)$  in qualche ordine. Allora il codice lineare  $n = 2^m - 1, k = 2^m - 1 - m$  su  $F_2$  la cui parity-check matrix è esattamente  $H$  è chiamato un **Hamming Code** binario di lunghezza  $2^m - 1$ .

**DOMANDA: COME VERIFICARE CHE UN CODICE È PERFETTO?**

Basta verificare che abbia distanza minima uguale a **3**, ovvero che riesca almeno a correggere **1 errore per codice**.

Possiamo anche verificare che sia un **Hamming Code**, dato che è un codice perfetto. Ovvero, dati  $n$ , il numero di colonne e  $k$ , il numero di righe della matrice  $H$ , un codice Hamming è un codice lineare  $n = 2^m - 1, k = 2^m - 1 - m$  grande  $2^m - 1$ .

## Remarks sugli Hamming Code

Vorrei puntualizzare due cose sugli Hamming codes:

1. Si noti che il syndrome decoding è facile facile: se l'errore è  $z = 0$ , allora la sindrome è  $s = 0$ , ma se  $w_H(z) = 1$ , allora  $s = c_i$ , la  $i$ -esima colonna di  $H$ . Dunque la sindrome identifica direttamente il luogo dell'errore.
2. Si ricordi che un codice  $C$  può correggere tutti i pattern di peso  $\leq 1$  sse le sfere di Hamming di radio 1 in  $V_n(F_2)$  contiene  $n + 1$  vettori, così che un single-error-correcting code può avere al più  $\frac{2^n}{n+1}$  codewords. In particolare se  $n = 2^m - 1$ , ci possono essere al massimo  $\frac{2^{2^m}-1}{2^m} = 2^{2^m-1-m}$  codewords, l'esatto numero dell'Hamming code!

## Esempio estratto dalla prova

**N. 4** — The output of a binary source is divided into sequences of  $k = 4$  bits and coded by a ECC with the following parity check matrix:

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

1. Determine the rate, the minimum distance, the error correction, and the error detection capabilities of this ECC.
2. Derive the matrix generator for this ECC.
3. Determine if this is a perfect code.

1. Determinare:

1. Rateo: per una matrice generatrice  $n \times k$  il rateo è  $k/n$ , qui la matrice generatrice è  $7 \times 4$  quindi il rateo è  $\frac{4}{7}$ .
2. Distanza minima: la distanza minima è il numero minimo di colonne linearmente dipendenti della matrice  $H$ . Ovvero il numero minimo di colonne che sommate (in binario) fa 0. In questo caso  $d_{\min} = 3$ .
3. L'ECC è un single-error-correcting code dato che rispetta la disequazione  $3 \geq 2 * e + 1$  Dove  $e = 1$ .

2.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

3. Si tratta di un  $n \times k$  Hamming code dove  $n = 2^m - 1 = 7$  e  $k = 2^m - 1 - m = 4$  e quindi  $m = 3$ .