INF2310 – Digital bildebehandling

FORELESNING 11

KOMPRESJON OG KODING – I

Andreas Kleppe

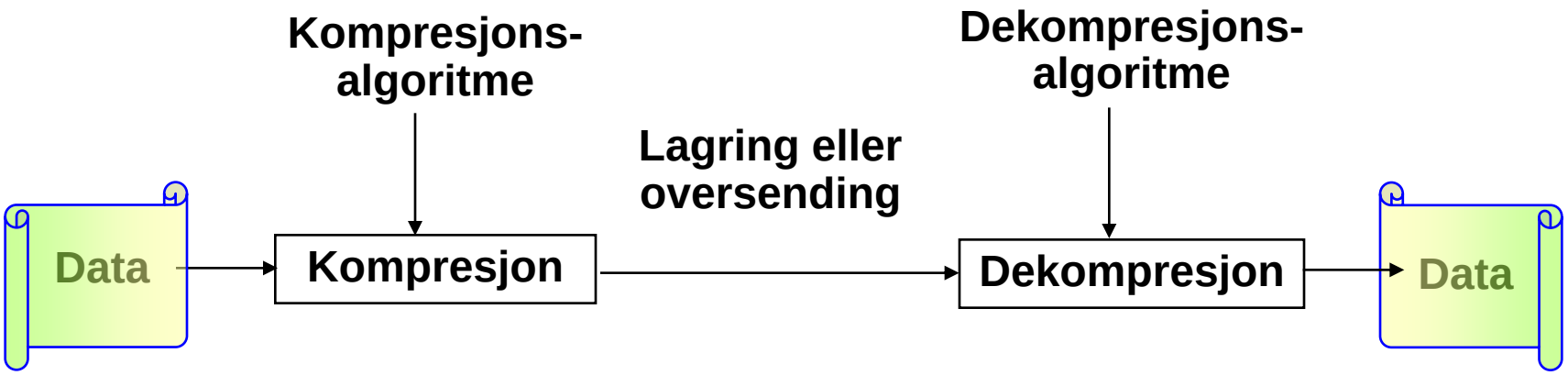
Tre steg i kompresjon
Redundanser
Transformer
Koding og entropi
Shannon-Fano og Huffman

Kompendium: Frem t.o.m. 18.7.2 + Appendiks B

Anvendelser

- Kompresjon og koding benyttes for å **redusere antall biter** som skal til for å beskrive bildet (eller en god tilnærming av bildet).
- En mengde anvendelser innen datalagring og dataoverføring.
  - Televideokonferanser
  - Fjernanalyse / meteorologi
  - Overvåking / fjernkontroll
  - Telemedisin / medisinske arkiver (PACS)
  - Dokumenthåndtering / FAX
  - Multimedia / nettverkskommunikasjon
  - Mobil kommunikasjon
  - MP3-spillere, DAB-radio, digitalkameraer, …
- **Tidsforbruket er viktig**, men det varierer om man ønsker å minimere kompresjonstiden eller dekompresjonstiden.
  - Det man gjør oftest ønsker man at tar kortest tid. Asymmetrisk kompresjon?
  - Begge tidene kan være omtrent like viktige. Symmetrisk kompresjon?

Noen begreper

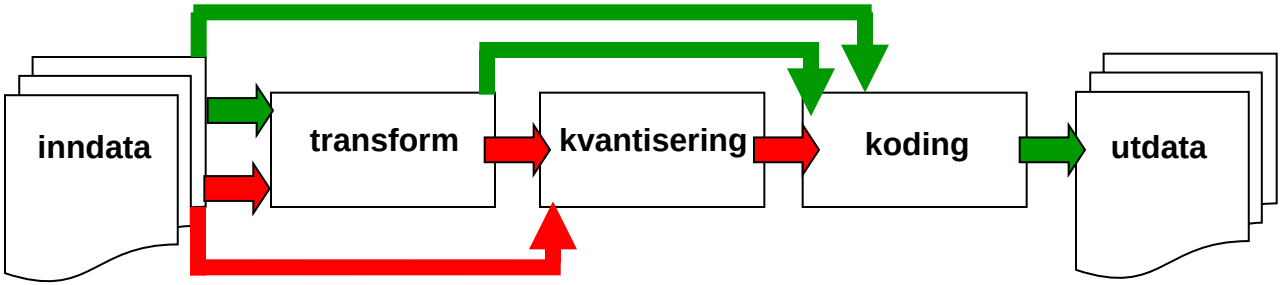


- **Bildekompresjon** består i å pakke informasjonsinnholdet i bildet ved å ikke lagre redundant informasjon.
  - Bildet **komprimeres** og deretter lagres eller overføres dataene.
  - Når bildet senere skal brukes, så **dekomprimeres** dataene.

- Koding er en del av kompresjon, men vi koder for å lagre effektivt, ikke for å hemmeligholde eller skjule informasjon.

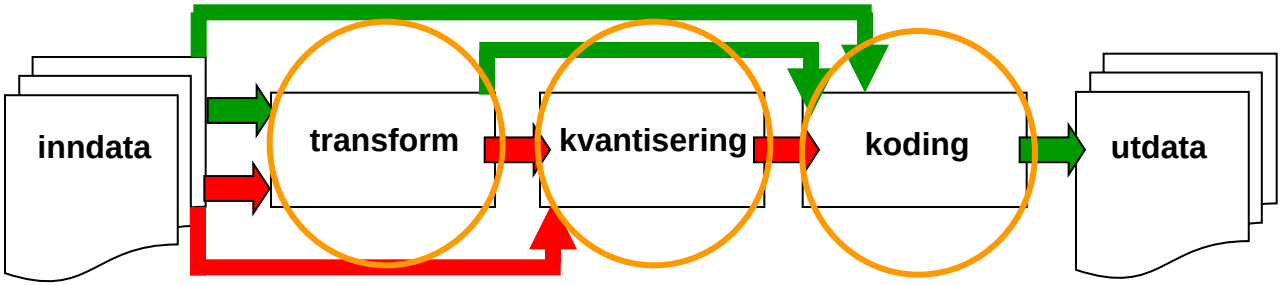
Kompresjon

- Kompresjon kan deles inn i tre steg:
  - **Transform** - representer bildet mer kompakt.
  - **Kvantisering** - avrunding av representasjonen.
  - **Koding** - produksjon og bruk av kodebok.



- Kompresjon kan gjøres:
  - **Eksakt / tapsfri** (eng.: *lossless*) – følg de grønne pilene.
    - Her kan vi rekonstruere den originale bildet eksakt.
  - **Ikke-tapsfri** (eng.: *lossy*) – følg de røde pilene.
    - Her kan vi ikke rekonstruere bildet eksakt.
    - Resultatet kan likevel være «godt nok».
  - Det finnes en mengde ulike metoder for begge kategorier.

De tre stegene i kompresjon



- Mange kompresjonsmetoder er basert på å **representere** bildet på en annen måte, altså **transformer** av original-bildet.
  - Eks.: Differansetransform, løpelengde-transform.
- Hvis vi **kvantiserer** det (originale eller transformerte) bildet, så kan ikke dette reverseres ⇒ ikke-tapsfri kompresjon.
- Til slutt **koder** vi, d.v.s. transformerer melding til binærrepresentasjon.
  - Bygger ofte på normaliserte histogrammer.
- **Koding er alltid reversible.**
- **Transformene** vi bruker **er alltid reversible.**
- **Kvantisering er ikke reversibelt!**

Eksempler: Plassbehov uten kompresjon

- Digitalt RGB-bilde:
  - 512 x 512 x 8 biter x 3 farger = 6 291 456 biter
  - 3264 x 2448 x 8 biter x 3 farger = 191 766 528 biter ≈ 24 MB
- Røntgen-bilde:
  - 7 112 x 8 636 piksler à 12 biter: 737 030 784 biter
- Radarbilde fra Radarsat-1-satellitten:
  - 400 MB, 300 km × 300 km, 16 biter per piksel.
  - For miljøovervåkning av Middelhavet:
    28 slike bilder trengs for å dekke hele Middelhavet.
- 3D-seismikk-data fra 60 000 km^2 i Nordsjøen.
  - 4 000 GB = 4 TB (terabyte)

Plass og tid

- Digitale data kan ta stor plass.
  - Spesielt lyd, bilder og video.
- Eksempler:
  1. Digitalt bilde:

     512 x 512 x 8 biter x 3 farger = 6 291 456 biter
  2. Røntgenbilde:

     7112 x 8636 x 12 biter per piksel = 737 030 784 biter

- Overføring av data tar tid:

Linje med 64 kbit/sek:

1. **ca. 1 min. 38 s.**
2. **ca. 3 timer 12 min.**

Linje med 1 Mbit/sek:

1. **ca. 6 s.**
2. **ca. 12 min.**

Overføringskapasitet og bps

- Filstørrelser er oftest gitt i binære enheter, gjerne i potenser av 1 024:
  - Kibibyte (KiB = 2^{10} byte = 1 024 byte),
  - Mebibyte (MiB = 2^{20} byte = 1 048 576 byte),
  - Gibibyte (GiB = 2^{30} byte = 1 073 741 824 byte)
- Overføringshastigheter og linjekapasitet angis **alltid** i titallsystemet, oftest som antall biter per sekund:
  - 1 kbps = 1000 bps = 10^3 biter per sekund
  - 1 Mbps = 1000 kbps = 10^6 biter per sekund
  - 1 Gbps = 1000 Mbps = 10^9 biter per sekund
- Kapasitet for noen typer linjer:
  - GSM-telefonlinje: 9,6 kbps
  - ADSL (Asymmetric Digital Subscriber Line): Opptil 8 Mbps
  - ADSL2+: Opptil 24 Mbps

Melding, data og informasjon

- Vi skiller mellom meldingens data og informasjon:

- **Melding:** teksten eller bildet som vi skal lagre eller sende.
- **Data:** strømmen av biter som lagres på fil eller sendes.
- **Informasjon:** et matematisk begrep som kvantifiserer hvor overraskende / uventet en melding er.

  - Et varierende signal har mer informasjon enn et monotont signal.

  - I bilder har kanter rundt objekter høyt informasjonsinnhold, spesielt kanter med mye krumning.

Redundans

- Vi kan bruke ulike mengder data på samme melding.
  - Et bilde av et 5-tall (50 x 50 piksler à 8 biter = 20 000 bits).
  - Teksten «fem» i 8-biters ISO 8859-1 (24 biter).
  - ISO 8859-1 tegnet «5» (8 biter).
  - Et binært heltall «101» (3 biter).

- **Redundans**: Det som kan «fjernes» fra dataene uten å miste (relevant) informasjonen.
  - Med «fjerne» menes her å redusere plassen dataene tar.
  - Vi tar utgangspunkt i dataene for et ukomprimert bilde, d.v.s. alle pikslene lagres separat med naturlig binærkoding.

- I kompresjon ønsker vi å fjerne redundante biter.

Ulike typer redundans

- Psykovisuell redundans. ⟶ Mer generelt: **Irrelevant informasjon**: Unødvendig informasjon for anvendelsen, f.eks. for visuell betraktning av hele bildet.
  - Det finnes informasjon vi ikke kan se.
    - Enkle muligheter for å redusere redundansen: Subsample eller redusere antall biter per piksel.
- Interbilde-redundans.
  - Likhet mellom nabobilder i en tidssekvens.
    - Kode noen bilder i tidssekvensen og ellers bare differanser.
- Intersampel-redundans.
  - Likhet mellom nabopiksler.
    - Hver linje i bildet kan løpelengde-transformeres.
- Kodings-redundans.
  - Gjennomsnittlig kodelengde minus et teoretisk minimum.
    - Velg en metode som er ”grei” å bruke og gir liten kodingsredundans.

Kompresjonsrate og redundans

- **Kompresjonsraten:**

$$CR = \frac{i}{c}$$

der i er antall bit per sampel originalt,
og c er antall bit per sampel i det komprimerte bildet.

- **Relativ redundans:**

$$R = 1 - \frac{1}{CR} = 1 - \frac{c}{i}$$

- **«Percentage removed»:**

$$PR = 100\left(1 - \frac{c}{i}\right)\%$$

Hvor god er bildekvaliteten?

- Hvis vi bruker **ikke-tapsfri kompresjon**
  må vi kontrollere at kvaliteten på ut-bildet er «**god nok**».

- Betegn M×N inn-bildet for *f* og ut-bildet etter kompresjon og så dekompresjon for *g*. Feilen forårsaket av komprimeringen er da:

  e(x,y) = g(x,y) – f(x,y)

- RMS-avviket (kvadratfeilen) mellom bildene er:

$$e_{RMS} = \sqrt{\frac{1}{MN}\sum_{x=1}^{M}\sum_{y=1}^{N} e^2(x,y)}$$

- Vi kan betrakte feilen som **støy** og
  se på midlet kvadratisk signal-støy-forhold (SNR):

$$(SNR)_{MS} = \frac{\sum_{x=1}^{M}\sum_{y=1}^{N} g^2(x,y)}{\sum_{x=1}^{M}\sum_{y=1}^{N} e^2(x,y)}$$

Hvor god er bildekvaliteten?

- RMS-verdien av SNR er da:

$$(SNR)_{RMS} = \sqrt{\frac{\sum_{x=1}^{M}\sum_{y=1}^{N} g^2(x,y)}{\sum_{x=1}^{M}\sum_{y=1}^{N} e^2(x,y)}}$$

- Bildekvalitetsmålene ovenfor slår sammen alle feil over hele bildet.
  - Vårt synssystem er ikke slik!
    - F.eks. vil mange små avvik kunne føre til en større $(SNR)_{RMS}$ enn enkelte manglende eller falske objekter i forgrunnen, men vi vil oppfatte at sistnevnte er av dårligst kvalitet.

- Ofte ønsker vi at bildekvalitetsmålet skal gjenspeile **vår oppfatning av bildets kvalitet**.
  - F.eks. hvis formålet med bildet er visuell betraktning.
  - Husk: Vår oppfatning er subjektiv!

Hvor god er bildekvaliteten?

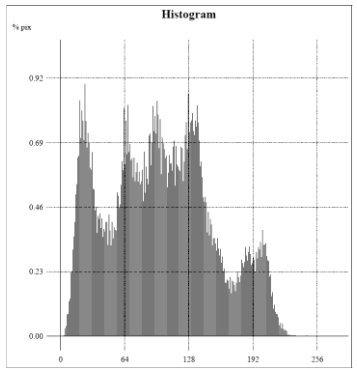
- Et bildekvalitetsmål som godt gjenspeiler vår oppfatning vil typisk basere seg på flere parametre.
  - Hver parameter prøver å indikere hvor ille vi oppfatter en side ved kompresjonsfeilen.
  - Bildekvalitetsmålet er én verdi som baserer seg på alle parametrene.
- Feil rundt kanter oppfattes som ille.
- Feil i forgrunnen oppfattes som verre enn feil i bakgrunnen.
- Manglende eller falske strukturer oppfattes som ille.
- **Kompresjonsgraden** bør trolig **variere rundt i bildet**:
  - Komprimer nesten-homogene områder kraftig og godta noe feil.
    - Husk fra Fourier-analysen: 2D DFT-en har få ikke-null-koeffisienter.
  - Komprimer kanter og linjer mindre og godta mindre feil.
    - Husk fra Fourier-analysen: En skarp kant bidrar til mange ikke-null-koeffisienter i 2D DFT-en.

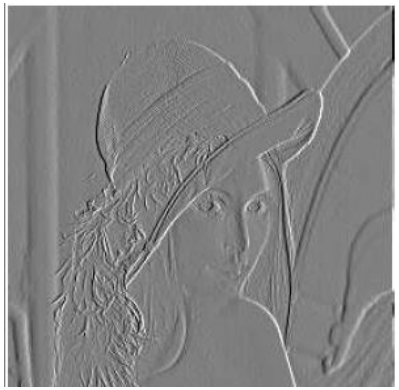
Differansetransform

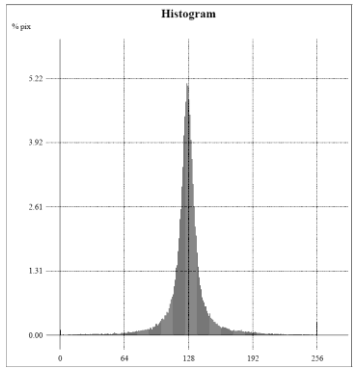
- Gitt en rad i bildet med gråtoner:
  $f_1,.....f_N$ der $0 \le f_i \le 2^b-1$
- Transformer (reversibelt) til

$g_1 = f_1$, $g_2 = f_2-f_1$, ... , $g_N = f_N-f_{N-1}$

- Merk at: $-(2^b-1) \le g_i \le 2^b-1$
- Trenger derfor b+1 biter per g_i hvis vi skal tilordne like lange kodeord til alle mulig verdier.
- I differansehistogrammet vil de fleste verdiene samle seg rundt 0.
- Naturlig binærkoding av differansene er ikke optimal.

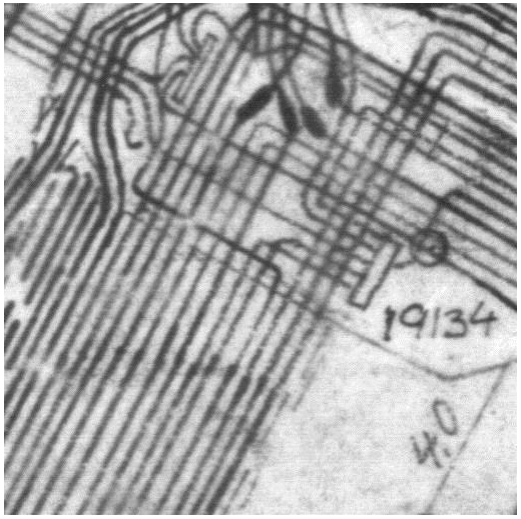




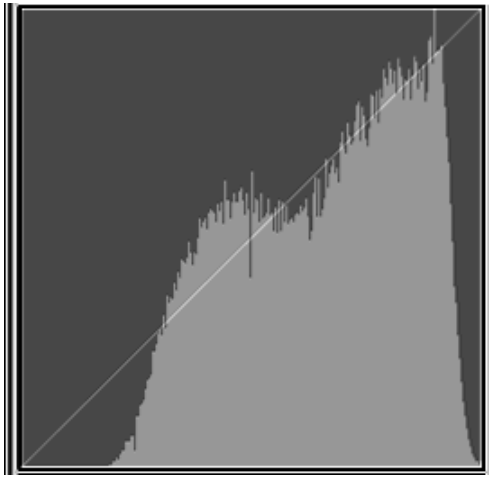


Differansebilder og histogram

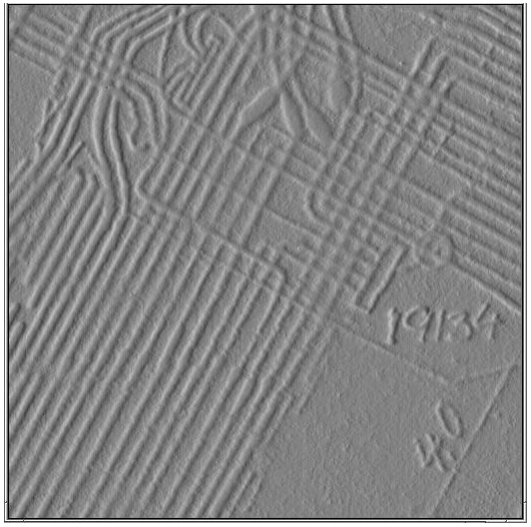
Original:



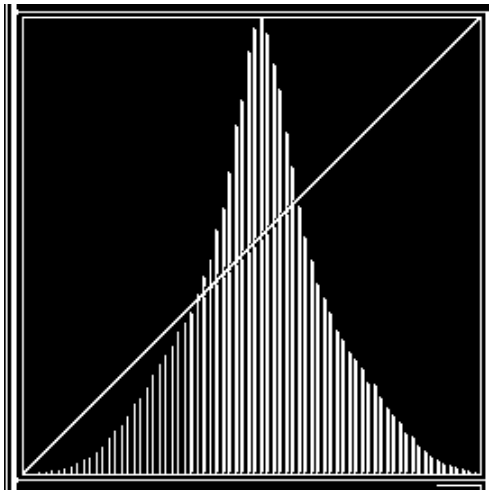
Originalt histogram:



Differansebilde:



Histogram til differansebildet:

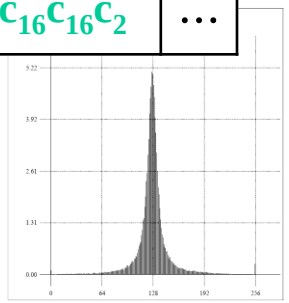


Mulig koding av differansetransform

- Ta 16-ords naturlig binærkode; c_1=0000, c_2=0001, …, c_{16}=1111
- Tilordne de 14 kodeordene i midten, c_2, c_3, …, c_{15} til differansene -7, -6, …, -1, 0, 1, 2, …, 5, 6.
- Bruk kodene c_1 og $\mathbf{c_{16}}$ til å indikere om differansen $\Delta x < -7$ eller om $\mathbf{\Delta x \geq 7}$ (to-sidet shift-kode)
  - Hvis kodeordet starter c_1 er intervallet skiftet med -14; -21, -20, …, -15, -14, -13, -12, …, -9, -8
  - Eksempel: $\Delta x = -22 \Rightarrow c_1c_1c_{15}$ (-14 + (-14) + 6)
  - Eksempel: $\Delta x = 21 \Rightarrow c_{16}c_{16}c_2$ (14 + 14 + (-7))

…	-22	-21	…	-8	-7	…	0	…	6	7	…	20	21	…
…	$c_1c_1c_{15}$	c_1c_2	…	c_1c_{15}	c_2	…	c_9	…	c_{15}	$c_{16}c_2$	…	$c_{16}c_{15}$	$c_{16}c_{16}c_2$	…

- Kodeordets lengde øker trinnvis med 4 biter.
  - Neppe helt optimalt i forhold til differansehistogrammet.



Løpelengde-transform

- Ofte inneholder bildet objekter med lignende gråtoner, f.eks. svarte bokstaver på hvit bakgrunn.
- Løpelengde-transformen (eng.: *run-length transform*) prøver å utnytte at **nabopiksler på samme rad ofte er like**.
  - Kompresjonen blir dårlig hvis dette ikke er tilfelle i det aktuelle bildet.
  - Løpelengde-transformen blir mer effektiv ettersom kompleksiteten i bildet blir mindre.
- Løpelengde-transformen er reversibel.
- Hvis pikselverdiene til en rad er:

  333333555555555544777777 (24 tall)
- Så starter løpelengde-transformen fra venstre og finner tallet 3 gjentatt 6 ganger etter hverandre, og returnerer derfor tallparet (3,6). **Formatet er: (tall, løpelengde)**
- For hele sekvensen vil løpelengdetransformen gi de 4 tallparene:

  (3,6), (5,10), (4,2), (7,6) (merk at dette bare er 8 tall)
- Kodingen avgjør hvor mange biter vi bruker for å lagre tallene.

Bare løpelengder, ikke tall

- I binære bilder trenger vi bare å angi løpelengden for hvert «run».
  - Må også angi om raden starter med hvitt eller svart «run», alternativt forhåndsdefinere dette og tillate en «run length» på 0.

- Histogrammet av løpelengdene er ofte ikke flatt.
  - Bør derfor benytte da koding som gir korte kodeord til de hyppigste løpelengdene.

- I ITU-T (tidligere CCITT) standarden for dokument-overføring per fax så Huffman-kodes løpelengdene.
  - Forhåndsdefinerte Huffman-kodebøker, én for svarte og én for hvite ”runs”.

- Tall-spesifiseringen i løpelengde-transformen av et gråtonebilde kan med fordel fjernes dersom ett tall forekommer *svært* hyppig.

Koding

- Et ***alfabet*** er mengden av alle mulige symboler (f.eks. gråtoner).
- Ofte får hvert symbol får et ***kodeord***.
- Tilsammen utgjør kodeordene ***kodeboken***.

- Koding skal være **reversibel**.
  - Denne egenskaper ved koder kalles for **unik dekodbarhet**; en sekvens kodeord kan dekodes på én og bare én måte.
  - Hvis hvert symbol har et kodeord betyr dette at kodeordet skal entydig gi det originale symbolet.

- **Instantant dekodbare koder** kan dekodes uten skilletegn.

Naturlig binærkoding

- Alle kodeord er like lange.
- Eks: En 3-biters kode gir 8 mulige verdier:

Symbolnr.	1	2	3	4	5	6	7	8
Symbol	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
Kode c_i	000	001	010	011	100	101	110	111

- Naturlig binærkoding er bare «optimal» hvis alle verdiene i sekvensen er like sannsynlige.
  - Med «optimal» menes her at kompresjonen er «best» mulig.

"Gray code"

Er den konvensjonelle binære representasjonen av gråtoner «robust»?

- Anta at vi har et gråtonebilde med b bitplan.
- Vi ønsker minst mulig kompleksitet i hvert bitplan.
  - Hvis vi antar at verdiene til nabopiksler er omtrent like, vil mindre kompleksitet i hvert bitplan bety at bitene i den alternative binærrepresentasjonen avviker mindre mellom nærliggende verdier.
  - Lav kompleksitet i hvert bitplan gjør at løpelengde-transform av hvert bitplan gir bedre komprimering.
- Konvensjonell binærrepresentasjon gir ofte høy bitplan-kompleksitet.
  - Hvis gråtoneverdien fluktuerer mellom 2^k-1 og 2^k vil k+1 biter skifte verdi. Eks.: 127 = 01111111 og 128 = 10000000
- I «Gray code» **skifter** alltid bare **én bit** når gråtonen endres med 1.
- Overgangen fra naturlig binærkode til «Gray code» er en transform, men både naturlig binærkode og «Gray code» er koder.
  - Både i naturlig binærkode og i «Gray code» er alle kodeord er like lange. Så forskjellen er bare hvilke kodeord som tilordnes hvilke symboler.

"Gray code"-transformasjoner

- Transformasjon fra naturlig binærkode (BC) til Gray-kode (GC):
  1. Start med MSB i BC og behold alle 0 inntil du treffer 1.
  2. 1 beholdes, men alle følgende bits komplementeres inntil du treffer 0.
  3. 0 komplementeres, men alle følgende bit beholdes inntil du treffer 1.
  4. Gå til 2.

Huskeregel: **Marker forskjellene** fra forrige bit. Bruk at før første bit er 0.

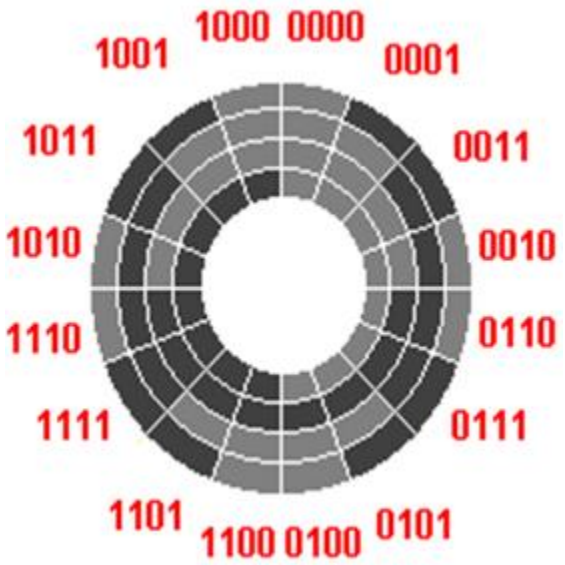
- Fra Gray-kode til naturlig binærkode:
  1. Start med MSB i GC og behold alle 0 inntil du treffer 1.
  2. 1 beholdes, men alle følgende bits komplementeres inntil du treffer 1.
  3. 1 komplementeres, men alle følgende bits beholdes inntil du treffer 1.
  4. Gå til 2.

Huskeregel: **Fyll inn mellom par av 1-ere** og fjern den siste 1-eren i hvert par. Hvis antall 1-ere er odde; fyll inn fra siste 1-er.

Eksempel: Gray-koding

4-biters Gray- og naturlig binærkode:

Gray-kode	Naturlig binærk.	Desimal-tall
0000g	0000b	0d
0001	0001	1
0011	0010	2
0010	0011	3
0110	0100	4
0111	0101	5
0101	0110	6
0100	0111	7
1100	1000	8
1101	1001	9
1111	1010	10
1110	1011	11
1010	1100	12
1011	1101	13
1001	1110	14
1000	1111	15

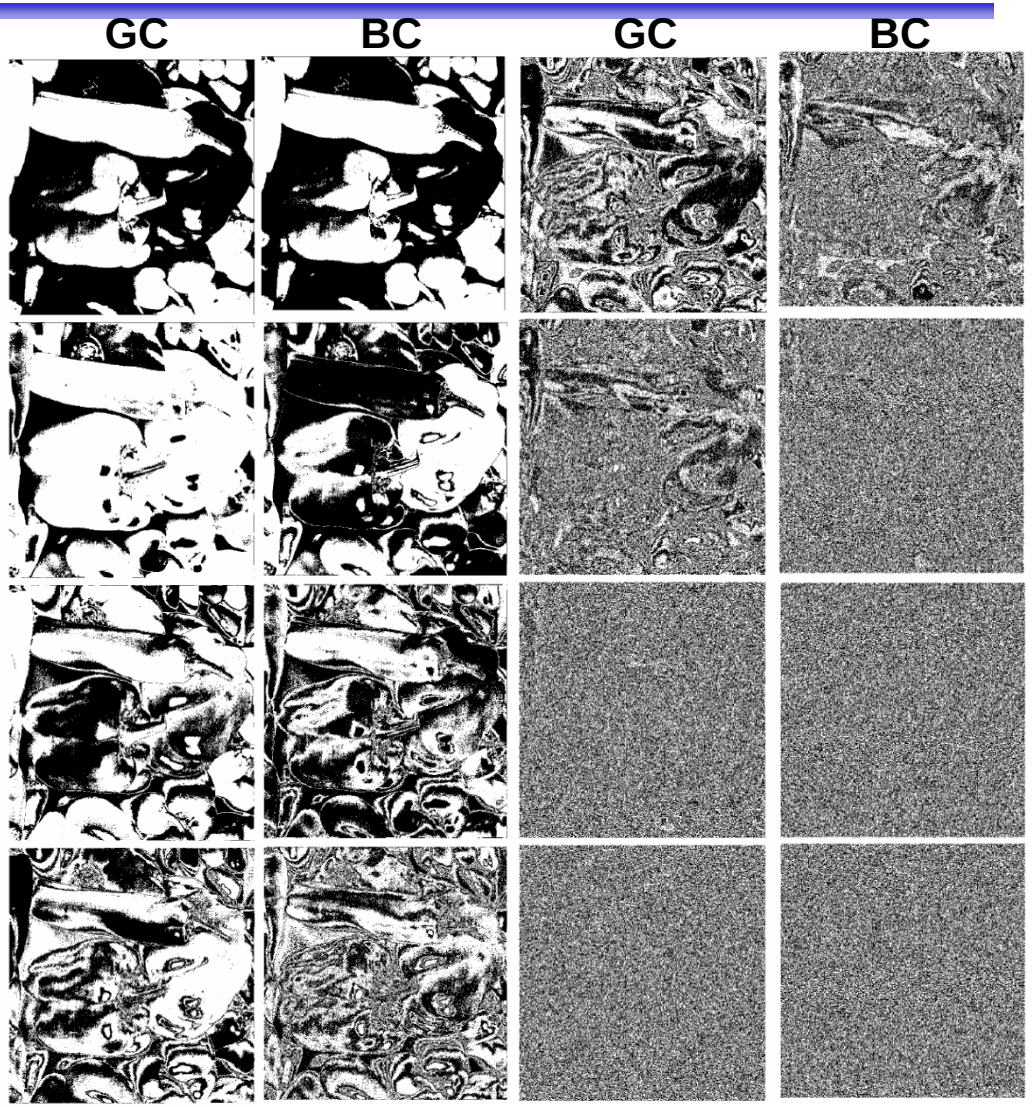


«Gray code shaft encoder»
Brukes for sikker avlesing av vinkel, f.eks. i styring av robot-armer.

Koden patentert av Gray i 1953, men ble brukt i Emilie Baudot's telegrafkode fra 1870.

Gray-kode i gråtonebilder

- MSB er alltid lik i de to representasjonene.
- Større homogene områder i hvert bitplan i Gray-kode enn i naturlig binærkode.
- Flere bitplan med «støy» i naturlig binærkode.
- => Løpelengdetransform av hvert bitplan gir bedre kompresjon v.b.a. Gray-kode enn naturlig binærkode.



Informasjonsteori og koding

- Koding bygger på sannsynligheter.
- Forekommer en pikselverdi oftere enn en annen, bør førstnevnte lagres med mindre antall biter for å bruke minst mulig biter på å lage hele bildet.
- Det er altså plassbesvarende å bruke flere biter på symboler som forekommer sjeldent, fordi hyppige symboler da kan bruke færre biter.
- **Vi vil bruke et variabelt antall biter per symbol.**
- Vi skal nå se på koding av enkeltpiksler.
- Interpiksel redundans minimeres av transform-steget:
  - Eks.: Differanse-transform, løpelengde-transform.

Koder med variabel lengde

- Når symbolene forekommer med ulik sannsynlighet er det bedre å bruke **kodeord med variabel lengde**.
  - Hyppige symboler ⇨ kortere kodeord.
  - Sjeldne symboler ⇨ lengre kodeord.
  - Dette var forretningsideen til Samuel Morse;

De vanligste symbolene i engelsk tekst er:

e, t, a, o, i, n, …

Morse-kode: . og mellomrom varer 1 enhet, - varer 3 enheter

A	. -	F	. . - .	K	- . -	P	. - - .	U	. . -
B	- . . .	G	- - .	L	. - . .	Q	- - . -	V	. . . -
C	- . - .	H	. . . .	M	- -	R	. - .	W	. - -
D	- . .	I	. .	N	- .	S	. . .	X	- . . -
E	.	J	. - - -	O	- - -	T	-	Y	- . - -

Entropi: En liten forsmak

- Entropi er et matematisk mål på informasjonsmengden i en sekvens med symboler (f.eks. tegn eller gråtoner).
  - Strengt tatt er entropi informasjonen i en tilfeldig variabel.
  - Når vi snakker om entropien av en sekvens med symboler så mener vi entropien av den diskrete variabelen der sannsynligheten til hvert symbol er dets frekvens i sekvensen.
    - D.v.s. at bare frekvensene til symbolene brukes, ikke posisjonene.
    - Hvis man antar at pikselverdiene er uavhengige realisasjoner av en underliggende diskret variabel, er entropien i sekvensen et estimat på entropien i variabelen.
- Entropi angir gjennomsnittlig informasjon per symbol.
  - Når vi bare ser på symbolenes frekvenser, ikke posisjoner.
- Intuitivt har vi at en mindre sannsynlig hendelse (symbol) gir mer informasjon enn en mer sannsynlig hendelse (symbol).
  - Informasjon er relatert til mengden overraskelser.

Histogram og normalisert histogram

- Anta vi har en sekvens med N symboler.

- Tell opp antall ganger symbol s_i forekommer og la n_i være dette antallet.
  - Dette er det samme som histogrammet til sekvensen.

- Sannsynligheten til symbolene finnes da som:
  - $p_i = n_i / N$
  - Dette er det normaliserte histogrammet.
  - Hvis man antar at pikselverdiene er uavhengige realisasjoner av en underliggende diskret variabel, er p_i et estimat på sannsynligheten for at variabelen er symbol s_i.

Gjennomsnittlig antall biter per piksel

- Vi konstruerer en kode c_1, …, c_N
  slik at symbol s_i kodes med kodeordet c_i.
- b_i er lengden av kodeordet c_i (angitt i biter).
- Gjennomsnittlig antall biter per symbol for koden er:

$$R = b_1 p_1 + b_2 p_2 + \ldots + b_N p_N = \sum_{i=1}^{N} b_i p_i$$

- Entropien H gir en nedre grense for
  hvor mange biter vi gjennomsnittlig trenger per symbol
  (hvis vi bare koder ett symbol av gangen).

Informasjonsinnhold

- Definer informasjonsinnholdet $I(s_i)$ i hendelsen s_i ved:

$$I(s_i) = \log_2 \frac{1}{p(s_i)}$$

- $\log_2(x)$ er 2-er-logaritmen til x.
  - Hvis $\log_2(x) = b$ så er $x=2^b$
    - Eks: $\log_2(64)=6$ fordi $64=2^6$ (=2*2*2*2*2*2)
      $\log_2(8)=3$ fordi $8=2*2*2=2^3$
  - $\log_2$ (tall) = $\log_{10}$(tall) / $\log_{10}(2)$

- $\log_2(1/p(s_i))$ gir oss informasjonsinnholdet i hendelsen: «symbolet s_i forekommer en gang», uttrykt i biter.

Entropi

- Tar vi gjennomsnittet over alle symbolene s_i i alfabetet, får vi gjennomsnittlig informasjon per symbol.

$$H = \sum_{i=0}^{2^b-1} p(s_i) I(s_i) = -\sum_{i=0}^{2^b-1} p(s_i) \log_2(p(s_i))$$

- H er entropien til sekvensen av symbolene.

- **Entropien setter en nedre grense for hvor kompakt sekvensen kan representeres.**
  - Gjelder bare hvis vi koder hvert symbol for seg.

Øvre og nedre grense for entropi

- Hvis alle symboler like sannsynlige => entropi lik antall biter.
  - Hvis det er 2^b mulige symboler i alfabetet, og sannsynligheten for hvert av dem er $p(s_i) = 1/2^b$, så er entropien:

$$H = -\sum_{i=0}^{2^b-1} \frac{1}{2^b} \log_2(\frac{1}{2^b}) = -\log_2(\frac{1}{2^b}) = b$$

  - Altså: Hvis det er 2^b symboler som alle er like sannsynlige, så kan de ikke representeres mer kompakt enn med b biter per symbol.

- Hvis alle pikslene er like => entropi lik 0.
  - Hvis bare ett symbol forekommer, er sannsynligheten for dette symbolet lik 1, og alle andre sannsynligheter er lik 0. Siden $\log_2(1) = 0$ vil entropien da bli 0.

Entropi i et binært bilde: To eksempler

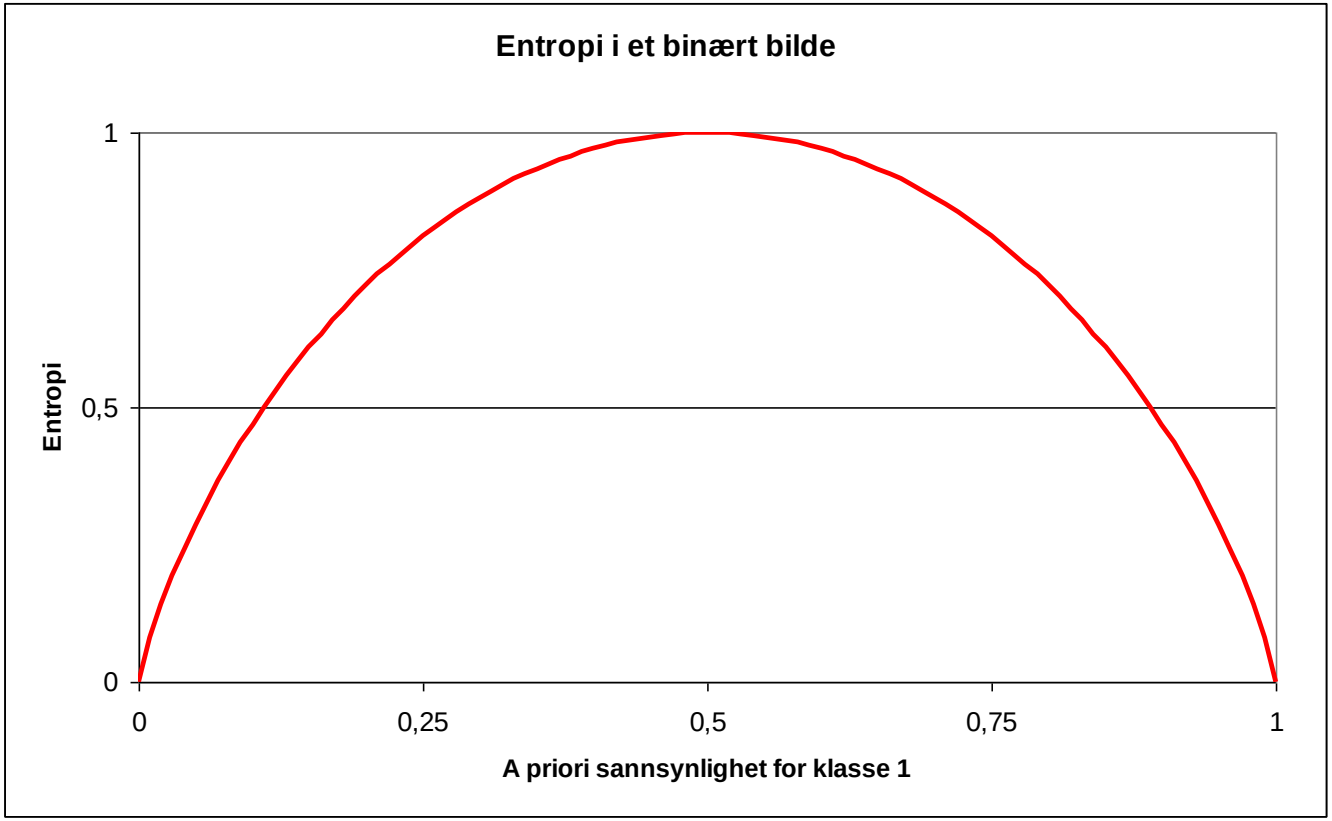
- Anta vi har et M×N binært bilde.
- Hvis vi skal lagre hver pikselverdi for seg selv, må vi alltid bruke MN biter, men hvor mye informasjon er det i bildet?
- Hvis det er like mange 0 som 1 i bildet, så er det like stor sannsynlighet for at neste piksel er 0 som 1. Informasjonsinnholdet i hver mulig hendelse er da like stort, derfor er entropien 1 bit.

$$H = \frac{1}{2}\log_2(\frac{1}{1/2}) + \frac{1}{2}\log_2(\frac{1}{1/2}) = \frac{1}{2} * 1 + \frac{1}{2} * 1 = 1$$

- Hvis det er 3 ganger så mange 1 som 0 i bildet, så er det mindre overraskende å få en 1, og det skjer oftere. Entropien er da mindre:

$$H = \frac{1}{4}\log_2(\frac{1}{1/4}) + \frac{3}{4}\log_2(\frac{1}{3/4}) = \frac{1}{4} * 2 + \frac{3}{4} * 0{,}415 = 0{,}5 + 0{,}311 = 0{,}811$$

Entropi i et binært bilde



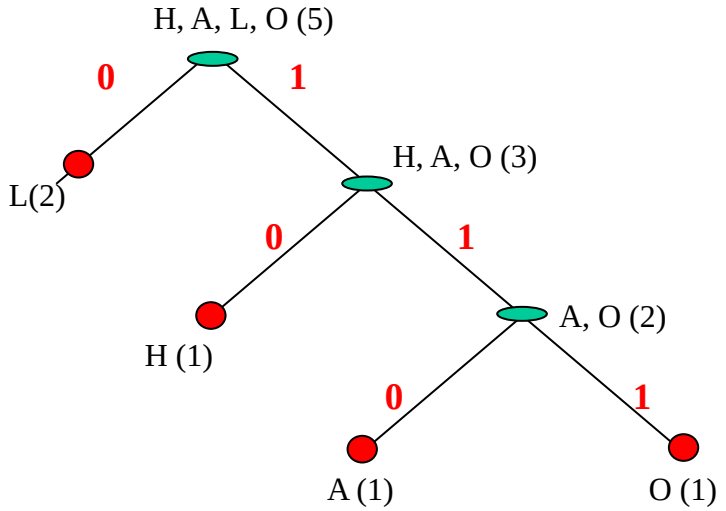
- Når vi lagrer pikselverdi for pikselverdi vil vi alltid måtte bruke 1 bit per piksel i et binært bilde, selv når entropien er nær 0!
- Kodingsredundansen er null når det er like mange svarte og hvite piksler.

Shannon-Fano-koding

En enkel metode:

1. Sorter symbolene etter hyppighet, hyppigst til venstre.
2. Del symbolene rekursivt i to grupper som forekommer så like hyppig som mulig.
   - Oppdelingen skal skje ved at symbolene til venstre for en grense blir en subgruppe, mens resten blir en annen subgruppe.
   - Venstre gruppe tilordnes 0, høyre gruppe tilordnes 1.
   - Rekursjonen stopper når hver gruppe inneholder ett symbol.
3. Traverser treet fra rot til hvert blad for å finne koden for hvert symbol.

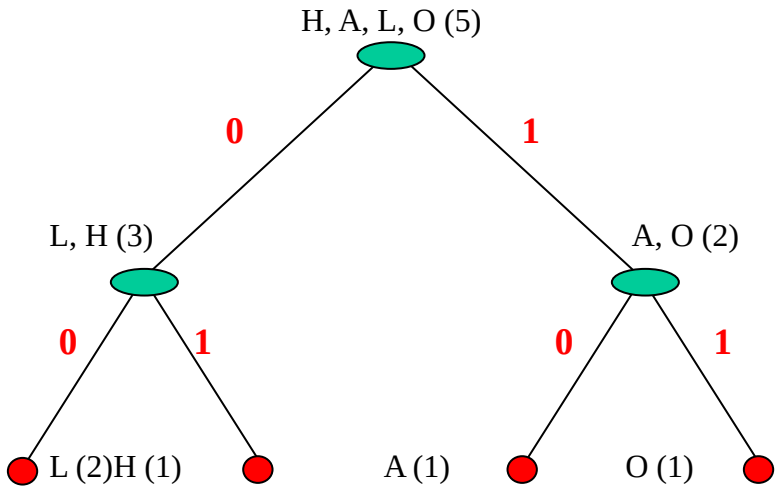
Eksempel: Koding av: HALLO



Symbol	Ant.	Kodeord	Lengde	Antall biter
L	2	0	1	2
H	1	10	2	2
A	1	110	3	3
O	1	111	3	3
Totalt antall biter				10

Shannon-Fano-koding

- Oppdeling i to «så like store grupper som mulig» kan gi et annet binærtre for eksempel: HALLO
  - Selv om treet er annerledes, og kodeboken blir forskjellig, så er koden unikt dekodbar.
  - For dette eksempelet er de to løsningene er likeverdige, men slik er det ikke alltid.



- Generelt for Shannon-Fano-koding er gjennomsnittlig antall biter per symbol er relatert til entropien:

$$H \leq R \leq H+1$$

- Øvre grense for kodingsredundans:

1 bit per symbol

Symbol	Ant.	Kodeord	Lengde	Antall biter
L	**2**	**00**	**2**	**4**
H	**1**	**01**	**2**	**2**
A	**1**	**10**	**2**	**2**
O	**1**	**11**	**2**	**2**
Totalt antall biter				**10**

Huffman-koding

- Huffman-koding er en algoritme for variabel-lengde koding som er **optimal** under begrensningen at vi **koder symbol for symbol**.
  - Med *optimal* menes her minst mulig kodings-redundans.

- Antar at vi kjenner hyppigheten for hvert symbol.
  - Enten spesifisert som en modell.
    - Huffman-koden er da optimal hvis modellen stemmer.
  - Eller så kan vi bruke symbol-histogrammet til sekvensen.
    - Huffman-koden er da optimal for sekvensen.
    - Ofte bruker vi sannsynlighetene i stedet, men vi kunne likegodt benyttet hyppighetene.

Huffman-koding: Algoritmen

Gitt en sekvens med N symboler:

1. Sorter symbolene etter sannsynlighet,
   slik at de minst sannsynlige kommer sist.
2. Slå sammen de to minst sannsynlige symbolene til
   en gruppe, og sorter igjen etter sannsynlighet.
3. Gjenta 2 til det bare er to grupper igjen.
4. Gi koden 0 til den ene gruppen
   og koden 1 til den andre.
5. Traverser innover i begge gruppene og
   legg til 0 og 1 bakerst i kodeordet
   til hver av de to undergruppene.

Eksempel: Huffman-koding

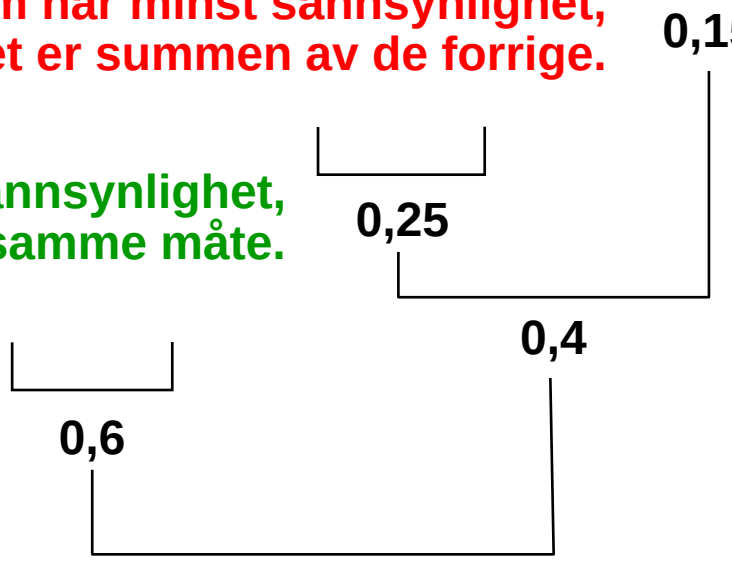
- La oss finne Huffman-koden til modellen som består av følgende seks begivenheter med sannsynligheter:

Begivenhet	A	B	C	D	E	F
Sannsynlighet	0,3	0,3	0,13	0,12	0,1	0,05

Slå sammen de to gruppene som har minst sannsynlighet, Den nye gruppens sannsynlighet er summen av de forrige.

Finn de to som nå har minst sannsynlighet, og slå dem sammen på samme måte.

Fortsett til det er bare to igjen.

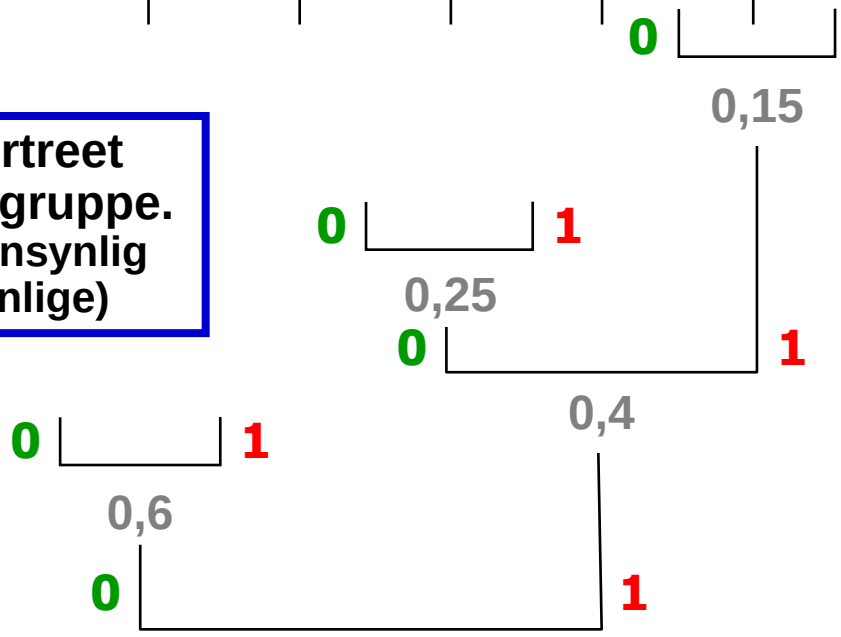


Eksempel: Huffman-koding

- La oss finne Huffman-koden til modellen som består av følgende seks begivenheter med sannsynligheter:

Begivenhet	A	B	C	D	E	F
Sannsynlighet	0,3	0,3	0,13	0,12	0,1	0,05

Gå baklengs gjennom binærtreet og tilordne 0 eller 1 til hver gruppe. (F. eks. kode 0 til den mest sannsynlig og kode 1 til den minst sannsynlige)



Eksempel: Huffman-koding

- Vi får dermed følgende kodebok:

Begivenhet	A	B	C	D	E	F
Huffman-kodeord	00	01	100	101	110	111

- Siden sannsynlighetene er:

Sannsynlighet	0,3	0,3	0,13	0,12	0,1	0,05

blir gjennomsnittlig antall biter per symbol:

$$R = b_1 p_1 + b_2 p_2 + \ldots + b_N p_N = \sum_{i=1}^{N} b_i p_i = 0{,}6 * 2 + 0{,}4 * 3 = 2{,}4$$

- Entropien H er her litt mindre enn R:

$$H = -\sum_{i=0}^{2^b - 1} p(s_i) \log_2 (p(s_i)) \approx 2{,}34$$

Huffman-koding: Kodingsredundans

- Shannon-Fano-koder har en øvre grensen for kodingsredundans på 1 bit per symbol.

- Kodingsredundansen til Huffman-koder har samme øvre grense, men har også en tettere grense dersom p_0 , sannsynligheten til det hyppigste symbolet, ikke er veldig stor.

$$R - H \leq p_0 + \log_2\left(\frac{2\log_2 e}{e}\right) \approx p_0 + 0{,}086$$

- Kodingsredundansen til Huffman-koder blir større ettersom p_0 nærmer seg 1; selv om p_0 er mye større enn 0,5 så må vi bruke 1 bit på å kode det tilhørende symbolet!

Generelt om Huffman-koding

- **Ingen kodeord danner prefiks i en annen kode.**
  - Dette sikrer at en sekvens av kodeord kan dekodes entydig og at man IKKE trenger endemarkører / skilletegn.
  - **Mottatt kodesekvens er unikt og instantant dekodbar.**
  - Dette gjelder også Shannon-Fano-koding og naturlig binærkoding.
- **Hyppige symboler har kortere kodeord enn sjeldne symboler.**
  - Dette gjelder også Shannon-Fano-koding.
- De to minst sannsynlige symbolene har like lange koder.
  - Siste bit skiller dem fra hverandre.
  - Dette gjelder også Shannon-Fano-koding.
- **Merk at kodeboken må overføres!**
  - Dette gjelder også Shannon-Fano-koding.
- **Lengden av en Huffman kodebok er opptil $G=2^b$ kodeord.**
- **Det lengste kodeordet kan ha opptil G-1 biter.**

Eksempel: Huffman-koding

- La oss Huffman-kode alfabetet som består av de seks mest sannsynlige symbolene i engelsk tekst:

Symbol	^	E	t	a	o	i
Sannsynlighet	0,34	0,19	0,14	0,12	0,11	0,10
Huffman-kodeord	00	10	010	011	110	111
Ordlengde	2	2	3	3	3	3
Entropi-bidrag	0,529	0,455	0,397	0,367	0,350	0,332

- Det gjennomsnittlige antall biter per symbol, R, er gitt ved:

$$R = \sum_{i=1}^{N} b_i p_i = 2 \cdot 0{,}53 + 3 \cdot 0{,}47 = 2{,}47$$

- Entropien H er litt mindre enn R: $H = -\sum_{i=0}^{2^b - 1} p(s_i) \log_2 (p(s_i)) \approx 2{,}43$

- Kodingsredundansen R-H er dermed: $R - H \approx 2{,}47 - 2{,}43 = 0{,}04$

Ideell og faktisk kodeord-lengde

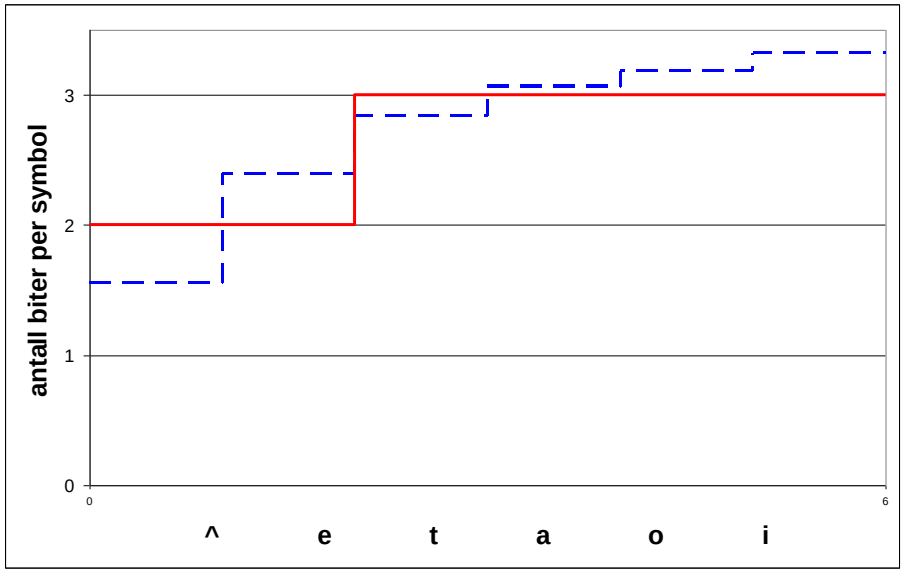
- Vi definerte informasjonsinnholdet $I(s_i)$ i hendelsen s_i ved:

$$I(s_i) = \log_2 \frac{1}{p(s_i)}$$

- Den ideelle binære kodeordlengden for symbol s_i er dermed:

$$b_i = -\log_2(p(s_i))$$

- Plotter den ideelle lengden på kodeordene (vist i rødt) sammen med de faktiske kodeordlengden (vist i blått) for forrige eksempel, får vi:



Når gir Huffman-koding ingen kodingsredundans?

- Den **ideelle binære kodeordlengden** for symbol s_i er:

$$b_i = -\log_2(p(s_i))$$

- Siden **bare heltalls ordlengder er mulig**,
  er det bare når $p(s_i) = \frac{1}{2^k}$
  for et heltall k som dette kan tilfredsstilles.

- Eksempel: Hvis sekvensen har sannsynlighetene:

Symbol	s_1	s_2	s_3	s_4	s_5	s_6
Sannsynlighet	0,5	0,25	0,125	0,0625	0,03125	0,03125
Huffman-kodeord	0	10	110	1110	11110	11111

så blir den gjennomsnittlige ordlengden R = 1,9375 = H.

Oppsummering

- Vi komprimerer for å redusere antall bits.
- Ved kompresjon fjernes/reduseres redundans:
  - Psykovisuell -, interbilde-, intersampel-, koding-redundans.
  - Kun når vi fjerner irrelevant informasjon for anvendelsen (f.eks. psykovisuell redundans) er kompresjonen ikke-tapsfri.
- Kompresjon kan deles inn i tre steg:
  1. Transform, f.eks. differanse- og løpelengde-transform.
  2. Kvantifisering. Hvis brukt, så blir kompresjonen ikke-tapsfri!
  3. Koding, f.eks. Shannon-Fano- eller Huffman-koding.
- Huffman-koding oppnår minst mulig kodingsredundans under antagelsen om at vi koder symbol for symbol.