INF 2310 – Digital bildebehandling 27.04.10

Kompresjon og koding – Del I

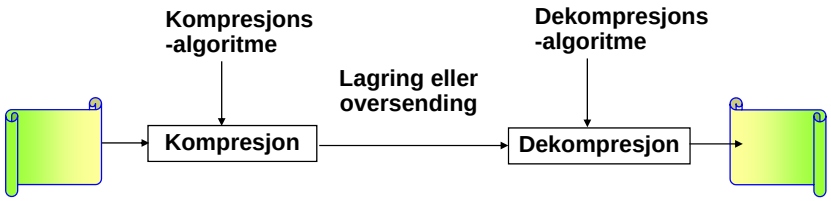
Tre steg i kompresjon
Redundans
Bildekvalitet
Transformer
Koding og entropi
Shannon-Fano og Huffman

GW: Kap. 8
unntatt 8.1.7, 8.2.2, 8.2.6, 8.2.10, 8.3

Anvendelser

- Kompresjon og koding benyttes for å redusere antall biter som skal til for å beskrive bildet (eller en god approksimasjon til bildet).
- En mengde anvendelser innen data-lagring og data-overføring
  - Televideo-konferanser
  - Fjernanalyse / meteorologi
  - Overvåking / fjernkontroll
  - Telemedisin / medisinske arkiver (PACS)
  - Dokumenthåndtering / FAX
  - Multimedia / nettverkskommunikasjon
  - Mobil kommunikasjon
  - MP3-spillere, DAB-radio, digitalkameraer, …
- **Tidsforbruket ved off-line kompresjon er ikke særlig viktig.**
- **Dekompresjons-tiden er langt viktigere.**
- **Ved "sanntids" data-overføring er tidsforbruket kritisk.**

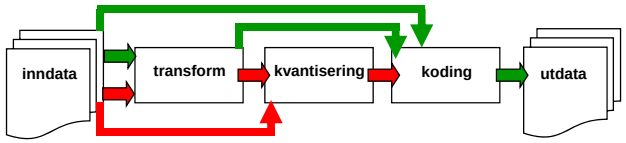
Noen begreper



- **Bildekompresjon** består i å pakke informasjonsinnholdet i bildet på en så kompakt måte at redundant informasjon ikke lagres.
- Dataene **komprimeres**, deretter lagres de.
- Når de senere skal leses, må vi **dekomprimere** dem.
- Koding er en del av kompresjon, men vi koder for å lagre effektivt, ikke for å hemmeligholde eller skjule informasjon.

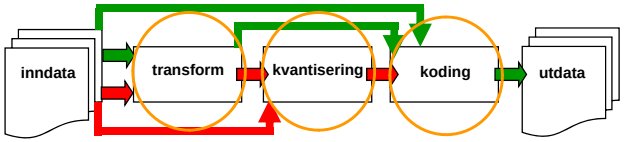
Kompresjon

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



- Kompresjons kan gjøres
  - **Eksakt / tapsfri** ("loss-less") – følg de grønne pilene
    - Her kan vi rekonstruere den originale meldingen eksakt.
  - **Ikke-tapsfri** ("lossy") – følg de røde pilene
    - Her kan vi ikke rekonstruere meldingen eksakt.
    - Resultatet kan likevel være "godt nok".
- Det finnes en mengde ulike metoder for begge kategorier kompresjon.

De tre stegene i kompresjon



- Mange kompresjons-metoder er basert på å representere dataene på en annen måte, altså **transformer** av original-dataene.
  - Differansetransform
  - løpelengder/run-length,
- Hvis vi **kvantiserer** original - dataene, så kan ikke dette reverseres.
- **Koding** bygger ofte på sannsynlighetsfordelinger,
  - Estimert ved normaliserte histogrammer.
- **Transformer og koding er alltid reversible.**
- **Kvantisering gir alltid et tap av presisjon.**

Eksempler - plassbehov

- SD 2timer video:
  - 30 (bilder/s)x 740 x 480 x 3 (RGB) x 3600 x 2 = 2.24 x 10^{14}
  - Tilsvarer 27 to-lags DVD-er ukomprimert
- Radar-bilde fra Radarsat-1 satellitten:
  - 400MB, 300 km × 300 km, 16 biter pr. piksel.
  - Miljøovervåkning av havområder:
    - Trenger 28 bilder for å dekke hele Middelhavet
- 3D-seismikk data fra 60.000 km^2 i Nordsjøen
  - 4 000 GB = 4 Terabyte...

Plass og tid

- Digitale data kan ta stor plass
  - Spesielt lyd, bilder og video
- Eksemper :
  1. Digitalt bilde:
     512 x 512 x 8 biter x 3 farger = 6 291 456 biter
  2. Røntgenbilde:
     7112 x 8636 x 12 biter pr. 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
  - Analogt modem: f.eks. 56 kbps
  - ADSL (Asymmetric Digital Subscriber Line): 1- 2 Mbps

Melding, data og informasjon

- Vi skiller mellom data og informasjon:

- **Melding:** teksten, bildet eller signalet som vi skal lagre.
- **Data:** strømmen av biter som lagres på fil eller sendes.
- **Informasjon:** Et matematisk begrep som kvantifiserer mengden overraskelse/uventethet i en melding.
  - Et varierende signal har mer informasjon enn et monotont signal.
  - I et bilde: kanter rundt objekter har høyest 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 a 8 biter = 20 000 bits)
  - Teksten "fem" i 8 bits ISO 8859-1 (24 biter)
  - ISO 8859-1 tegnet "5" (8 biter)
  - Et binært heltall "1 0 1" (3 biter)

- **Redundans** sier noe om hvor stor del av datamengden vi kan fjerne uten at vi mister informasjonen i dataene.

- Eks: vi skal lagre tallet 0, men hvordan lagres det faktisk:
  - Binært i en byte: 00000000 - 8 biter med 0
  - Standard 7 bits ASCII kode for 0
  - Vet vi at vi bare skal lagre tallet 0, trenger vi bare 1 bit.
  - Ved smart komprimering og dekomprimering kan vi fjerne redundante bits.

Ulike typer redundans

- Psykovisuell redundans
  - Det finnes informasjon vi ikke kan se.
    - Da kan vi sub-sample, eller redusere antall bit per piksel.
- Interbilde/temporal redundans
  - Det er en viss likhet mellom nabo-bilder i en tids-sekvens
    - Vi koder noen bilder i sekvensen, og deretter bare differanser.
- Intersampel/romlig redundans
  - Nabo-piksler ligner på hverandre eller er like.
    - Hver linje i bildet kan "run-length" transformeres.
- Kodings-redundans
  - Gjennomsnittlig kodelengde minus et teoretisk minimum.
    - Velg en metode som er "grei" å bruke, med liten redundans.

Kompresjonsrate og redundans

- **Kompresjonsraten :**

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

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

- **Relativ redundans :**

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

- **"percentage removed" :**

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

Bildekvalitet

- Hvis vi velger irreversibel kompresjon må vi kontrollere at kvaliteten på ut-bildet er "god nok".
- Gitt en N×M inn-bilde f(x,y) og et komprimert / dekomprimert ut-bilde g(x,y). Feilen vi gjør blir da:

  e(x,y) = g(x,y) – f(x,y)
- RMS-avviket (kvadratfeilen) mellom de to bildene blir da:

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

- Vi kan også betrakte feilen som "støy" og se på midlere kvadratisk signal-støy-forhold (SNR):

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

Bildekvalitet - 2

- RMS-verdien av SNR er da

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

- Kvalitetsmålene ovenfor slår sammen alle feil over hele bildet.
- Vårt synssystem er ikke slik !
- Fler-komponent kvalitetsmål er bedre! Del opp bildet etter lokal aktivitet!
- Fra Fourier-analyse:
  - For å representere en skarp kant kan vi trenge mange koeffisienter.
  - Homogene områder kan representeres ved få koeffisienter
- Kompresjonsgraden bør kanskje variere rundt i bildet:
  - Komprimer homogene områder kraftig.
  - Mindre kompresjon langs kanter og linjer, slik at detaljer beholdes.
- Mer om dette neste uke!

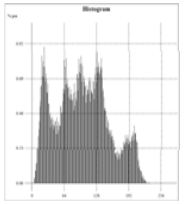
Kompresjon av bilder

- Det er en mengde redundant informasjon i bilder:
  - Mellom piksler på samme linje
  - Mellom ulike linjer i bildet
  - Objekter kan representeres mer kompakt enn med piksler.
  - Vi kan også ta i betraktning psykovisuelle effekter.
- Symmetrisk kompresjon: komprimering og dekomprimering tar like lang tid.
- Assymmetrisk kompresjon, komprimering og dekomprimering har ulik regnetid, ofte tillater man langsom komprimering, men ønsker rask dekomprimering.

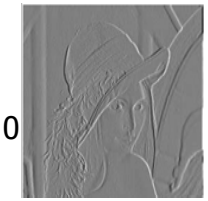
STEG 1: Transform Differansetransform

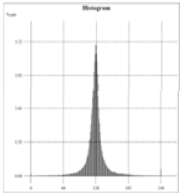
- Gitt en linje i bildet med gråtoner $f_1,.....f_N$, $0 \le f_i \le 2^b-1$.
- Transformer (reversibelt) til

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



- Vi trenger nå b+1 biter hvis vi skal tilordne like lange binære koder til alle mulig verdier.
- I differansehistogrammet vil de fleste verdiene samle seg rundt 0
- En naturlig bit-koding av differansene er ikke optimal.



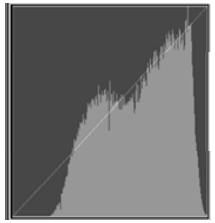


STEG 1: Transform

Differansebilder og histogram

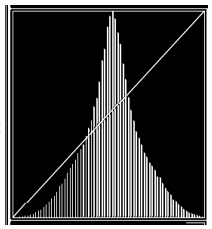
Original:

Originalt Histogram:



Differanse-bilde (linje for linje) :

Histogram til Differansebildet:



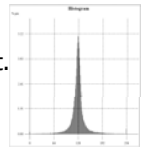
STEG 1: Transform

Litt mer om differansetransform

- Lag f.eks. en 16 ords naturlig kode
  - c_1=0000, c_2=0001,.... c_{16}=1111
- Tilordne de 14 kode-ordene i midten:
  - c_2,.... c_{15} til differansene -7, -6, ..., -1, 0, 1, 2, ..., 5, 6
- Kodene c_1 og c_{16} kan brukes til å indikere om differansen
  $\Delta x < -7$ eller om $\Delta x \geq 7$ (to-sidet shift-kode)
  - $\Delta x = -22 \Rightarrow c_1c_1c_{15}$ $\Delta x = 21 \Rightarrow c_{16}c_{16}c_2$

…	-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}	$c_{16}c_2$	…	$c_{16}c_{15}$	$c_{16}c_{16}c_2$	…

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



Huffman-koding etter differansetransform gir bedre resultat enn denne kodingen

STEG 1: Transform

Løpelengde-transform

- Ofte inneholder bildet objekter med lignende gråtoner,
  f.eks. svarte bokstaver på hvit bakgrunn.
- Vi kan benytte oss av kompresjonsteknikker
  som tar hensyn til at nabopiksler på samme linje ofte er like.
- Løpelengde-transform er reversibel.
- Først et eksempel:
  333333555555555544777777 (24 byte)
- Når tallet 3 forekommer 6 ganger etter hverandre,
  trenger vi bare lagre tallparet (3,6).
- Tilsammen trenger vi her 4 tallpar, (3,6), (5,10), (4,2), (7,6)
  altså 8 tall til å lagre hele sekvensen 24 tall ovenfor.
- Hvor mange biter vi bruker pr. tall, avhenger av den videre kodingen.

STEG 1: Transform

Løpelengder i binære bilder

- I to-nivå bilder trenger vi bare å angi løpelengden for hvert "run", forutsatt at vi vet om linjen starter med et hvitt eller et svart run.
- Vi trenger kodeord for EOL og EOI.
- Run-lengde histogrammet er ofte ikke flatt.
  - Benytter da en kode som gir
    korte kode-ord til de hyppigste run-lengdene.
- En standard (CCITT) Huffman kode basert på dokument-statistikk brukes for dokument-overføring pr fax.
- Egen kodebok for svarte og hvite runs.
- Mer effektiv dersom kompleksiteten i bildet er lav.

STEG 3: Koding

Koding

- Et *alfabet* er mengden av alle mulige symboler (gråtoner)
- Hvert symbol får et *kode-ord*.
- Tilsammen utgjør kodeordene *kode-boken.*

- Koding skal være reversibel
  - fra koden skal vi kunne rekonstruere det originale symbolet

- Unikt dekodbare koder har den egenskap at en mottatt sekvens av kode-ord kan dekodes på en og bare en måte.
- Instantant dekodbare koder kan dekodes uten skilletegn.

STEG 3: Koding

Naturlig binær-koding

- Alle kode-ord er like lange.
  - Kjenner vi noen eksempler på dette?
- Eks: En 3-biters kode gir 8 mulige verdier:

Symbol nr.	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

- Her er kodene bare en teller i symboltabellen
- Naturlig binærkoding er bare optimal hvis alle verdiene i sekvensen er like sannsynlige.

STEG 3: Koding

"Gray code"

Er den konvensjonelle binære representasjonen av gråtoner optimal?

- La oss se på et ett-bånds gråtone-bilde med b bit-plan.
- Ønskelig med minst mulig kompleksitet i hvert bit-plan
  - Da blir løpelengde-transformasjonen mest effektiv.

- Konvensjonell binær representasjon gir høy bit-plan kompleksitet.
  - Hvis gråtoneverdien fluktuerer mellom 2^k-1 og 2^k vil k+1 biter skifte verdi: eksempel: 127 = 01111111 mens 128 = 10000000

- I "Gray Code" skifter alltid bare en bit når gråtonen endres med 1.

- Overgangen fra binær kode til "gray code" er en transformasjon, men både naturlig binær kode og "gray code" er selvsagt koder.

STEG 3: Koding

Gray Code tranformasjoner

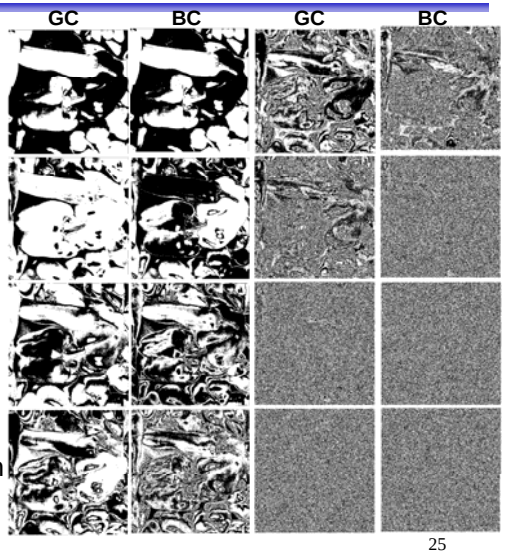
- Transformasjon fra "Binary Code" til "Gray Code":
  1. Start med MSB i BC og behold alle 0 inntil du treffer 1
  2. 1 beholdes, men alle følgende bits komplementeres inntil du treffer 0
  3. 0 komplementeres, men alle følgende bit beholdes inn du treffer 1
  4. Gå til 2.

- Fra "Gray Code" til "Binary Code":
  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

STEG 3: Koding

Gray kode i gråtonebilder

- Her ser vi alle bitplan for et 8-bits bilde.
- MSB (Most significant byte) er likt i de to representasjonene.
- Større homogene områder i hvert bitplan i Gray-kode enn i naturlig binærkode.
- Flere bitplan med støy i vanlig binærkode.
- Det er en gevinst i løpelengdekoding av bitplan i Gray kode.



Informasjonsteori og koding

- Kompresjon/koding bygger på sannsynligheter.
- Forekommer en pikselverdi ofte, bør vi lagre den med et lite antall biter for å bruke minst mulig lagerplass totalt.
- Et symbol som forekommer sjeldent, kan vi tillate oss å bruke mange biter på å lagre.
  - Vi bruker da et variabelt antall biter pr. symbol.
- Vi skal først se på koding av enkelt-piksler.
- Inter-piksel redundans minimeres av transform-steget:
  - Differanse-transform
  - Løpelengdetransform

Koder med variabel lengde

- For ulike sannsynligheter er **koder med variabel lengde** på kode-ordene bedre enn like lange koder
  - Hyppige symboler ⇨ kortere kode-ord.
  - Sjeldne symboler ⇨ lengere kode-ord.
  - Dette var forretnings-ideen til Samuel Morse

De vanligste symbolene i engelsk tekst er :

e, t, a, n, o, i,...

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 gjennomsnittlig informasjonsmengde i en sekvens av tegn eller tall.
- *Har vi en sekvens av N tall eller tegn som lagres med b biter pr. sampel, så kan vi si vi har et alfabet med 2^b mulige symboler.*
- Vi kan finne sannsynligheten for hvert symbol i alfabetet:
  - $P(s_i) = n_i/N$
- Vi er interessert i gjennomsnittlig informasjon pr. symbol.
- Intuitivt har vi at en mindre sannsynlig hendelse gir mer informasjon enn en mer sannsynlig hendelse
  - Informasjon er relatert til mengden redundans eller overraskelse.

Histogram og normalisert histogram

- 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.

Gjennomsnittlig antall biter pr. piksel

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

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

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

Informasjonsinnhold

- Entropi er relatert til hvor uventet en hendelse er:
  - Hyppige symboler er ikke uventet og har lav entropi.
- 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

- Hvis vi tar gjennomsnittet over alle symbolene s_i i alfabetet, får vi gjennomsnittlig informasjon pr. symbol.
- Entropi H:

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

- Entropien setter en nedre grense for hvor kompakt sekvensen kan representeres
  - Dette gjelder hvis vi bare koder hvert symbol for seg.
  - I praksis kan vi ikke kode et symbol med for eksempel2.3 bit så vi får kun en teoretisk nedre grense.

Øvre og nedre grense for entropi

- Hvis alle symboler like sannsynlige => entropi lik antall biter.
  - Det er 2^b symboler, og sannsynligheten for hvert av dem er $p(s_i)=1/2^b$. Da blir entropien

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

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

$$H = -\log_2(1) = 0$$

To eksempler

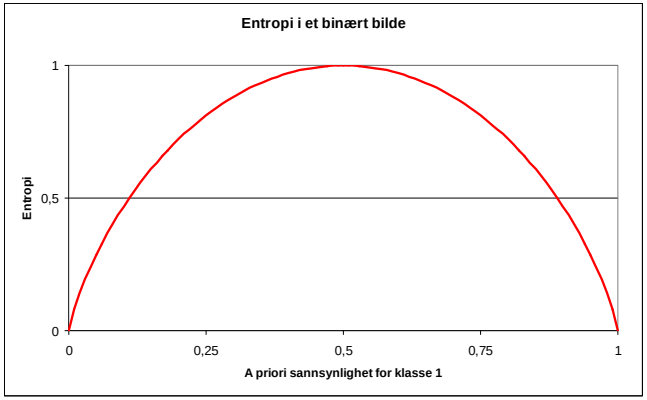
- Et binært bilde med N*M piksler inneholder 1 bit per piksel.
- Det er N*M biter data i bildet, men hvor mye informasjon er det i bildet?
- Hvis det er like mange 0 som 1 i bildet, så er det like stor sannsynlighet for at neste piksel er 0 som 1. Informasjonsinnholdet i hver mulig hendelse er da like stort, og entropien til et nytt symbol er 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:

$$H = \frac{1}{4}\log_2(\frac{1}{1/4}) + \frac{3}{4}\log_2(\frac{1}{3/4}) = \frac{1}{4}*2 + \frac{3}{4}*0.415 = 0.5 + 0.311 = 0.811$$

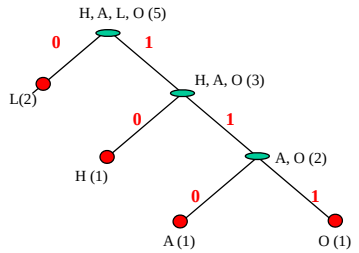
Entropi i binært bilde



- Vi vil alltid måtte bruke 1 bit per piksel i et binært bilde, selv om entropien godt kan bli nær null!
- Kodingsredundansen er null når det er like mange svarte og hvite piksler.

Shannon-Fano koding

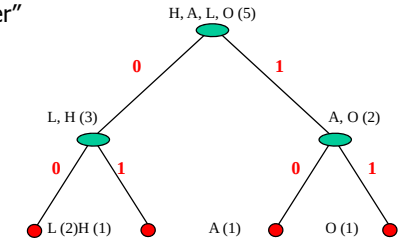
- En enkel metode:
  - Sorterer symbolene etter hyppighet.
  - Deler symbolene rekursivt i to "omtrent like store grupper".
    - Fortsett til hver gruppe er ett symbol.
  - Tilordner ett bit til hver gren i treet
    - 1 til høyre, 0 til venstre.
  - Traverser treet "fra rot til blad"
    - Finner koden for hvert symbol.
- Eksempel: "HALLO" :
- Koden er unikt dekodbar.



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 - II

- Oppdeling i "omtrent like store grupper" kan gi et annet binært tre:
- Samme eksempel: "HALLO"
- Selv om treet er annerledes, og kodeboken blir forskjellig, så er koden unikt dekodbar.



- Vi har altså flere likeverdige løsninger.

- 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 optimal koding med variabel-lengde koder.
- Huffman-koding er basert på at vi kjenner hyppigheten for hvert symbol
  - Dette betyr at vi må lage et histogram.
  - Ofte beregner vi sannsynlighetene
    - Men det holder at vi kjenner hyppighetene.
- Huffman-koden er unikt dekodbar.

Oppskrift - Huffman-koding

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 i en gruppe, og sorter igjen etter sannsynlighet.
3. Gjenta 2 til det bare er to grupper igjen.
4. Gi kodene 0 og 1 til de to gruppene.
   - Kode 0 til den mest og 1 til den minst sannsynlige av de to
5. Traverser bakover, og legg til 0 og 1 i kodeordet for de to minst sannsynlige gruppene i hvert steg.

Eksempel - Huffman-koding

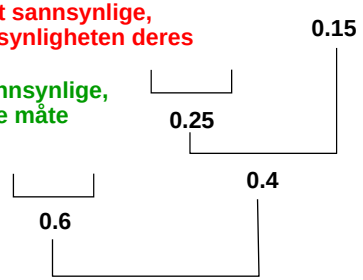
- Gitt 6 begivenheter A, B, C, D, E, F med sannsynligheter

Begivenhet	A	B	C	D	E	F
Sannsynlighet	0.3	0.3	0.13	0.12	0.1	0.05

Slå sammen de to minst sannsynlige,
Slå også sammen sannsynligheten deres

Finn de to som nå er minst sannsynlige,
og slå dem sammen på samme måte

Fortsett til det er bare to igjen

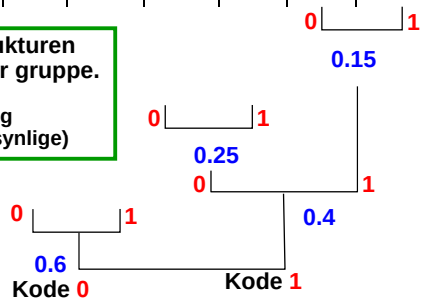


Eksempel - Huffman-koding

- Gitt 6 begivenheter A, B, C, D, E, F 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 strukturen og tilordne 0 eller 1 til hver gruppe.
(F. eks.
kode 0 til den mest sannsynlig
og kode 1 til den minst sannsynlige)



Kodeboken

- Dette gir følgende kodebok

Begivenhet	A	B	C	D	E	F
Kode	00	01	100	101	110	111

- Med sannsynlighetene 0.3 0.3 0.13 0.12 0.1 0.05
  blir gjennomsnittlig antall biter pr. symbol (R) for denne koden:
  (se foil 30)

$$R = b_1 p_1 + b_2 p_2 + \ldots + b_N p_N = \sum_{i=1}^{N} b_i p_i = 0.6 * 2 + 0.4 * 3 = 2.4$$

- Entropien H er her mindre enn R (se foil 32):

$$H = -\sum_{i=0}^{2^b - 1} p(s_i) \log_2(p(s_i)) = 2.34$$

Generelt om Huffman-koding

- **Ingen kode-ord danner prefiks i en annen kode**
  - Dette sikrer at en sekvens av kodeord kan dekodes entydig
  - man trenger IKKE ende-markører.
- **Mottatt kode er instantant (øyeblikkelig) dekodbar.**
  - Dette gjelder også Shannon-Fano og naturlig binærkoding.
- **Hyppige symboler har kortere koder enn sjeldne symboler.**
  - Dette gjelder også for Shannon-Fano.
  - De to minst sannsynlige symbolene har like lange koder,
    - siste bit skiller dem fra hverandre.
- **Merk at kodeboken må overføres!**
- **Lengden av en Huffman kodebok er G=2ᵇ kodeord.**
- **Det lengste kodeordet kan ha opptil G biter.**

Kodingsredundans

- La oss bruke de 6 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
kode	00	10	010	011	110	111
Ordlengde	2	2	3	3	3	3
entropi	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:

$$H = -\sum_{i=0}^{2^b - 1} p(s_i) \log_2(p(s_i)) = 2.34$$

- Kodingsredundansen R-H er dermed:

$$R - H = 2.47 - 2.34 = 0.13$$

Ideell og faktisk kodeord-lengde

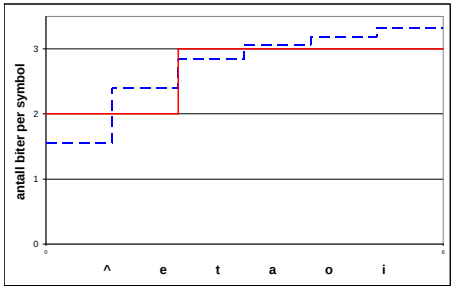
- Vi definerte informasjonsinnholdet I(s_i) i hendelsen s_i ved

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

- Den ideelle binære kodeordlengden for symbol s_i er

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

- Plotter den ideelle lengden på kodeordene sammen med den faktiske kodeordlengden, som er heltall. Stiplet linje er ideell kodelengde.



Når er Huffman-koding optimal?

- Den ideelle binære kode-ord lengden for symbol s_i er

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

- Siden bare heltalls ordlengder er mulig, er det bare

$$p(s_i) = \frac{1}{2^k}$$

for heltall k som tilfredsstiller dette.

Eksempel: hvis vi har

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
Kode	0	10	110	1110	11110	11111

blir den gjennomsnittlige ordlengden R = 1.9375 = H.

Kodingsredundans for Huffman

- Shannon-Fano metoden har en øvre grensen for kodingsredundans på 1 bit per symbol.
- Kodingsredundansen til Huffman-koden har en øvre grense som er en funksjon av p_0 , der p_0 er sannsynligheten til det hyppigste symbolet.
- Huffman-koden kan i enkelte tilfeller komme dårligere ut når det gjelder kodingsredundans enn Shannon-Fano, hvis p_0 er nær 1.