Norsk Regnesentral
NORWEGIAN COMPUTING CENTER

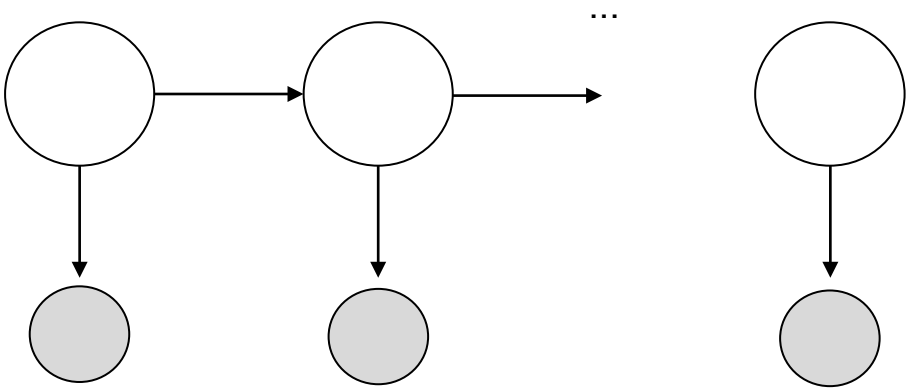
Sekvensmodeller (del 2)

Pierre Lison
plison@nr.no

IN2110: Språkteknologiske metoder

23. mars 2022

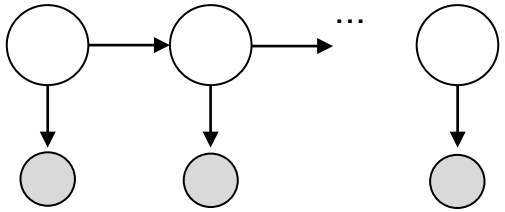
Dagens tema:



Modellering av sekvenser (del 2):
Hidden Markov Models
og litt om nevrale nettverk for sekvenser

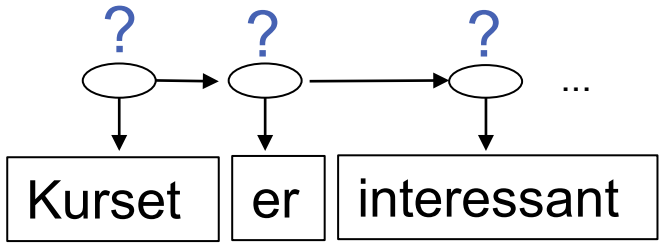
Hovedtema for i dag:

Sekvensmodeller, og mer spesielt *Hidden Markov Models*



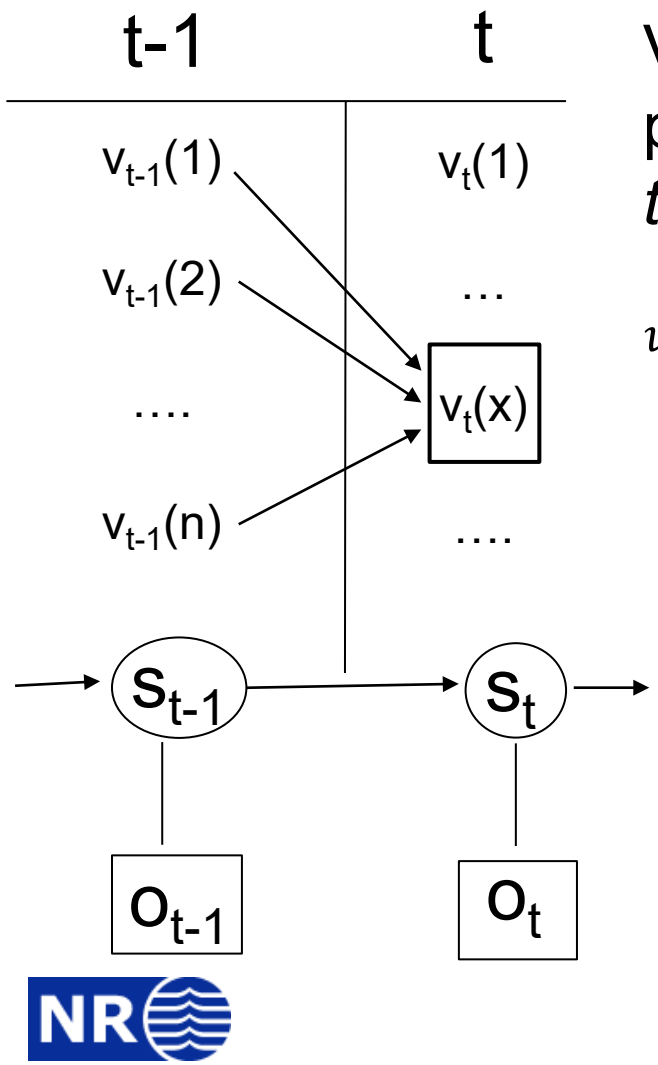
Modellering: hvordan behandler vi problemer som tar sekvensdata som input og output?

Dekoding: hvordan beregner vi den meste sannsynlige sekvensen av labeller (f.eks. POS tags) gitt en sekvens av observasjoner (f.eks. ord)?



Læring: Hvordan estimerer vi parametrene i HMMs fra (markerte) data?

Viterbi-dekoding



Verdien $v_t(x)$ er sannsynligheten for tag *x* på ordet i posisjon *t og den beste tagsekvensen* fram til *t-1:*

$$v_t(x) = \max_{1 \le y \le N} v_{t-1}(y)\, P(s_t = x | s_{t-1} = y) P(o_t | s_t = x)$$

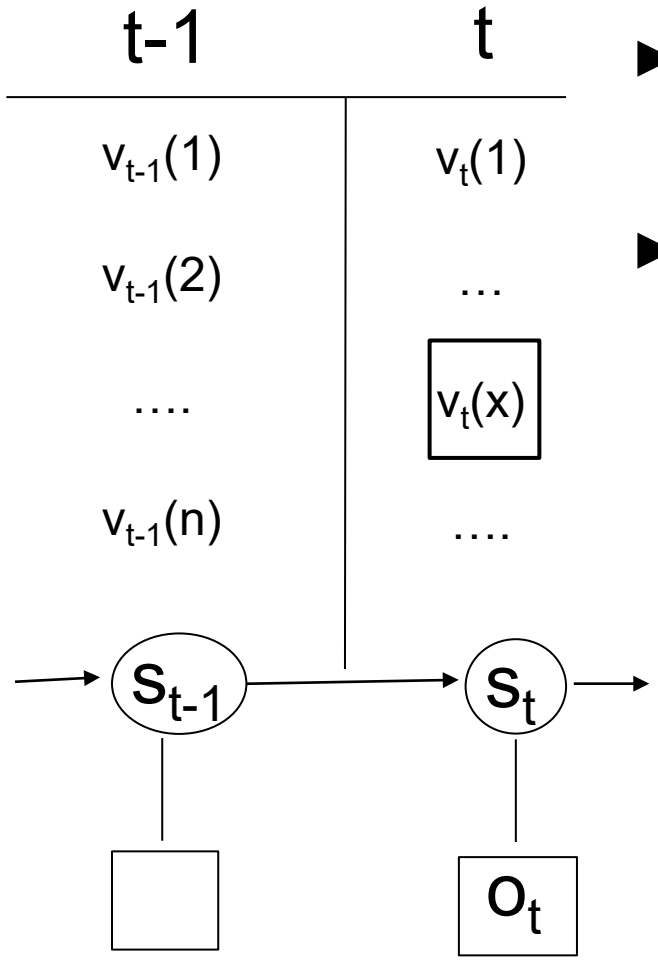
transisjonsmodell

emisjonsmodell

Vi tar kun hensyn til det *mest sannsynlige tag* y for t-1 (hvorfor?)

Verdien for tag *y* for det forrige ordet

Viterbi-dekoding



- Siste detalj: Viterbi-algorithme må også lagre såkalte **backpointers**
- Backpointer for $v_t(x)$ er tilstanden *y* i formelen vi nettopp har sett, dvs.

$$\max_{1 \le y \le N} v_{t-1}(y) P(s_t = x | s_{t-1} = y) P(o_t | s_t = x)$$

➔ Med disse backpointers kan vi lett ekstrahere de meste sannsynlige merkelappene når hele sekvensen er ferdig behandlet

Eksempel (fra boken)

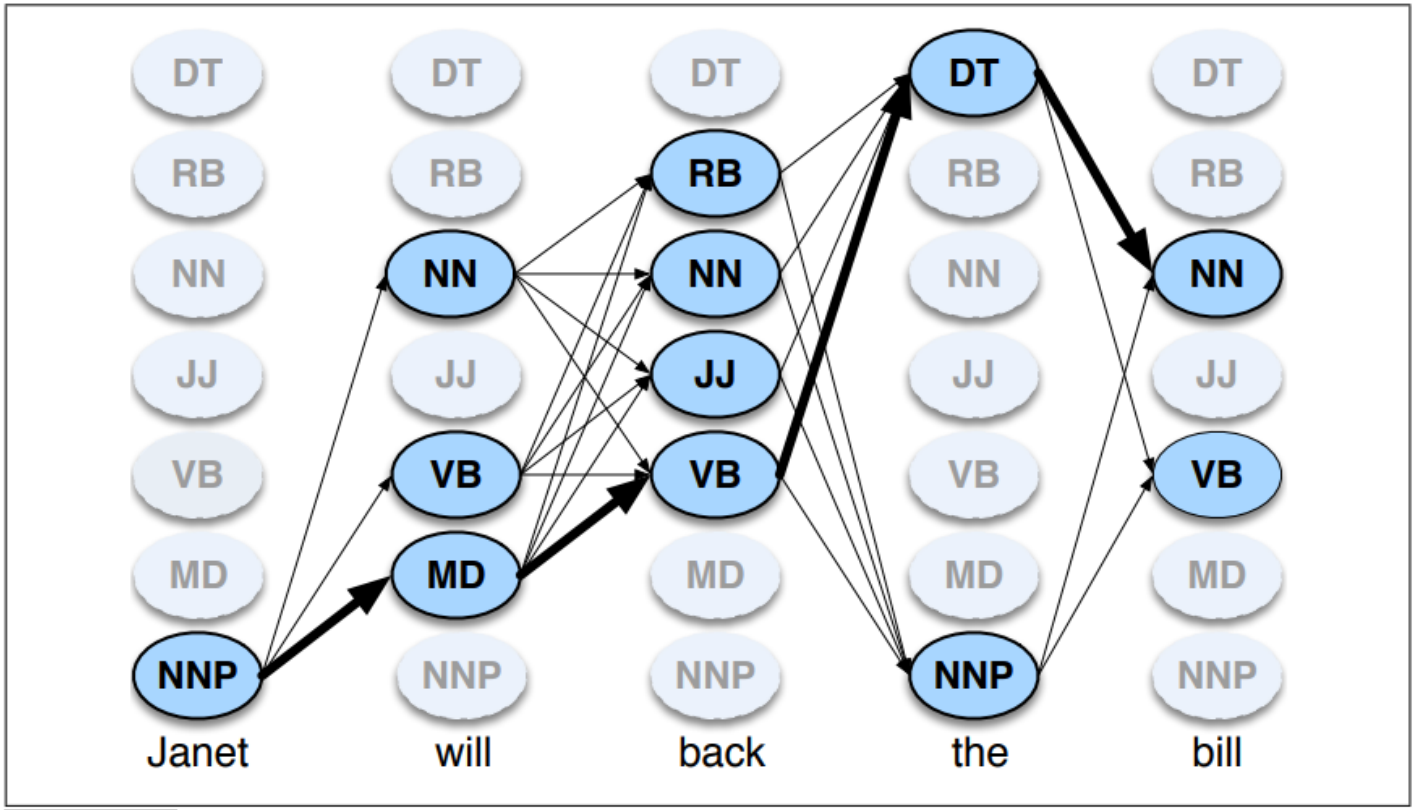


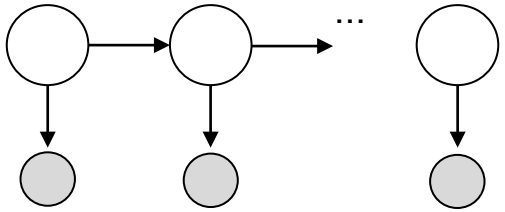
Figure 8.6 A sketch of the lattice for *Janet will back the bill*, showing the possible tags (q_i) for each word and highlighting the path corresponding to the correct tag sequence through the hidden states. States (parts of speech) which have a zero probability of generating a particular word according to the B matrix (such as the probability that a determiner DT will be realized as *Janet*) are greyed out.

Kommentarer

- Viterbi basert på "dynamic programming"
  - = vi bryter ned en komplisert beregning i mindre små beregninger som "gjenbruker" hverandres resultater (i vår tilfelle gjenbruker vi v_{t-1} for å beregne v_t)
  - *Edit distance* er et annet eksempel
- Viterbi likevel vanskelig å anvende på mer kompliserte NLP-oppgaver (slik som maskinoversettelse)
  - Long-range dependencies, stor antall mulig outputs
- Det finnes mange andre dekoding-algoritmer, blant annet **beam search** (som fokuserer på en beam av *k* hypoteser)

Hovedtema for i dag:

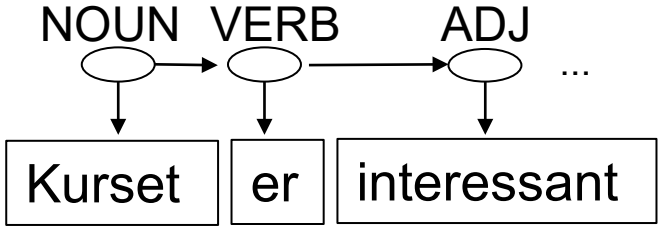
Sekvensmodeller, og mer spesielt *Hidden Markov Models*



Modellering: hvordan behandler vi problemer som tar sekvensdata som input og output?

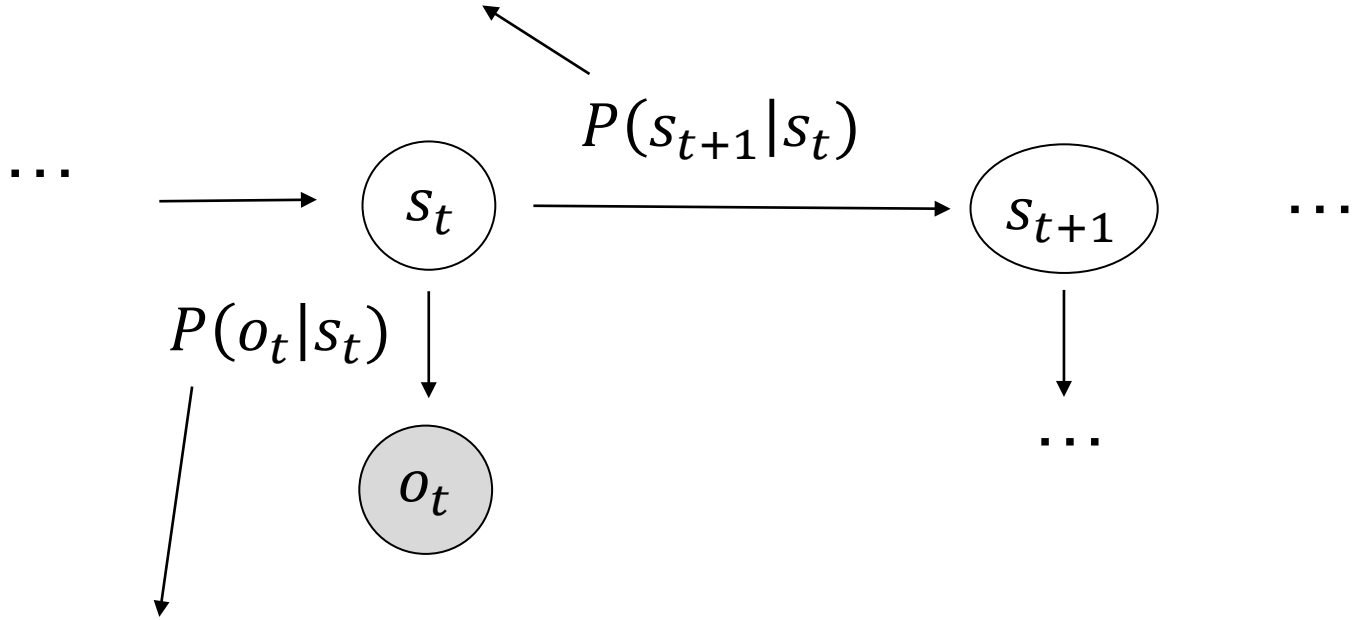
Dekoding: hvordan beregner vi den meste sansynnlige sekvensen av labeller (f.eks. POS tags) gitt en sekvens av observasjoner (f.eks. ord)?

Læring: Hvordan estimerer vi parametrene i HMMs fra (markerte) data?



Hidden Markov Models

Transisjonsmodell: hvor sannsynlig er det å bevege oss fra s_t til s_{t+1} ?



Emisjonsmodell: hvor sannsynlig er det å observere o_t hvis vi er i tilstanden s_t ?

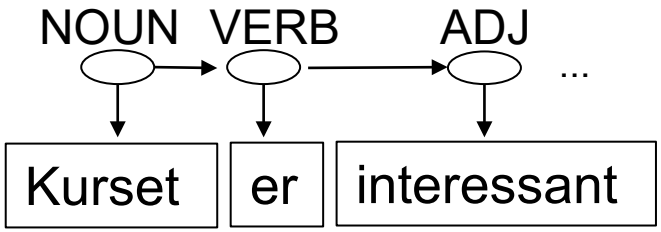
HMM-estimering

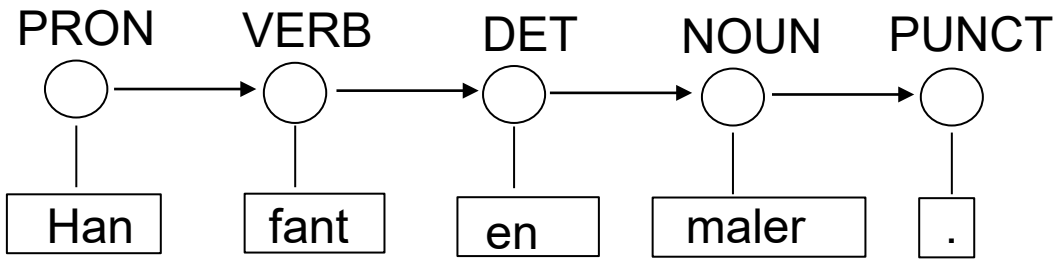
- Opp til nå har vi snakket om
  - Hvordan sekvensmodeller representeres matematisk
  - Hvordan vi effektivt beregner den mest sansynnlige outputsekvensen gitt inputsekvensen (decoding)
- Men hvordan kan vi *trene* disse sekvensmodellene? Kan vi bygge disse fra data?

Klart vi kan!

HMM-estimering

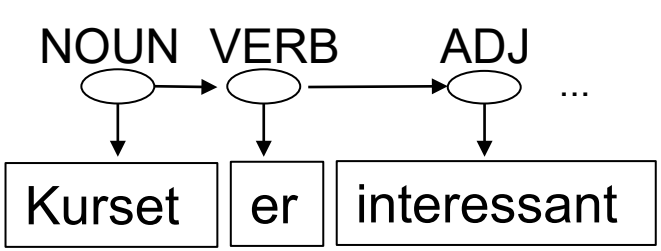
La oss anta at vi har et annotert treningsett

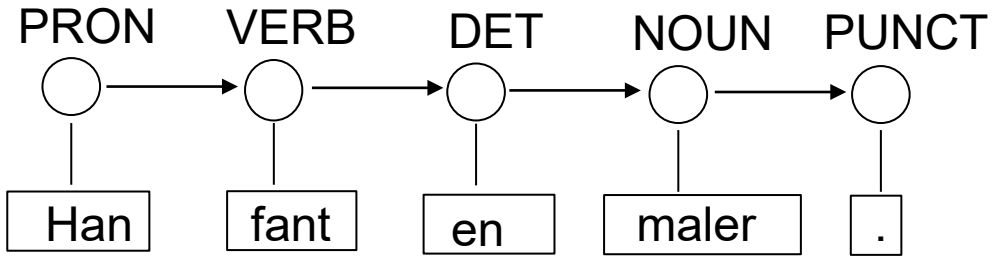




Vi har alt vi trenger for å estimere transisjonsmodellen og emisjonsmodellen!

HMM-estimering





Transisjonsmodell:

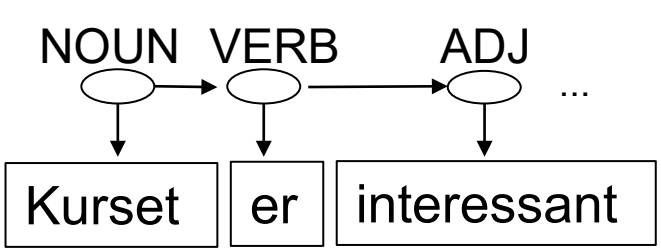
$$P(t_i \mid t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

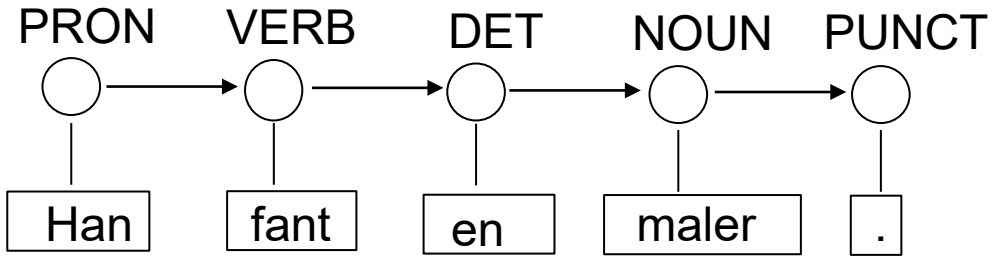
Antall ganger vi finner t_{i-1} fulgt av t_i i korpuset

Antall ganger vi finner merkelappen t_{i-1} i korpuset

Eksempel: P(t_i =ADJ | t_{i-1} = VERB) = 0.5

HMM-estimering





Emisjonsmodell:

$$P(w_i \mid t_i) = \frac{C(w_i, t_i)}{C(t_i)}$$

Antall ganger vi finner ordet w_i med merkelappen t_i i korpuset

Antall ganger vi finner merkelappen t_i i korpuset

Eksempel: P(w_i =maler | t_i = NOUN) = 0.5

Smoothing

- Hva med ordene som ikke forekommer i treningsett

$$P(w_i = bok \mid t_i = \text{NOUN}) = \frac{C(w_i{=}bok, t_i{=}NOUN)}{C(t_i{=}NOUN)} = 0\ ?$$

- Samme gjelder transisjoner fra t_{i-1} til t_i som ikke forekommer i treningsdata
- Ideelt sett bør disse sannsynlighetene være små, men > 0
  - Tar i betrakning at treningsett har en *begrenset størrelse*, og gir dermed imperfekte forekomsttallene

➔ **Smoothing** kan brukes til å korrigere sansynnlighetesfordelingene

Smoothing

- Laplace smoothing (også kalt “additive smoothing):

$$P(w_i \mid t_i) = \frac{C(w_i, t_i) + \alpha}{C(t_i) + \alpha V}$$

Hvor α er en parameter og V er antall unike ord i modellen

- Eksempel med α =0.01
  - $P(w_i = bok \mid t_i = NOUN) = \frac{0.01}{2+0.01*8} = 0.0048$
  - Og $P(w_i = maler \mid t_i = NOUN) = \frac{1+0.01}{2+0.01*8} = 0.4856$

Smoothing

- **Spørsmål**: hvorfor har vi αV i nevner?

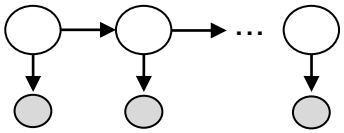
$$P(w_i \mid t_i) = \frac{C(w_i, t_i) + \alpha}{C(t_i) + \alpha V}$$

- Smoothing mest nyttig i emisjonsmodellen (gitt den store antall mulige observasjoner)
- Andre smoothingteknikker enn Laplace kan også brukes, blant annet *Kneser-Ney discounting* (se J&M)

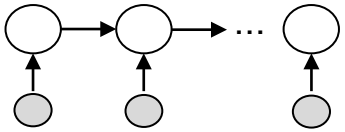
Andre sekvensmodeller

Mange ulike modeller for sekvensklassifisering:

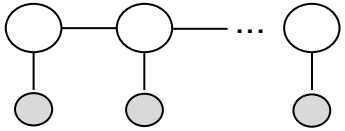
- Hidden Markov Models (HMMs)



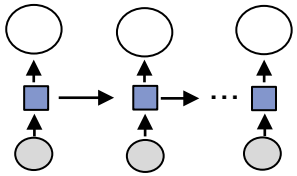
- Maximum Entropy Markov Models (MEMMs)



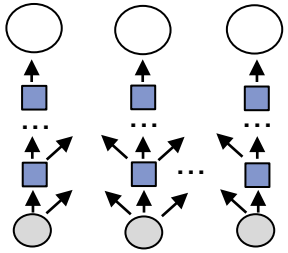
- Conditional Random Fields (CRFs)



- Recurrent Neural Networks (LSTMs, GRUs)



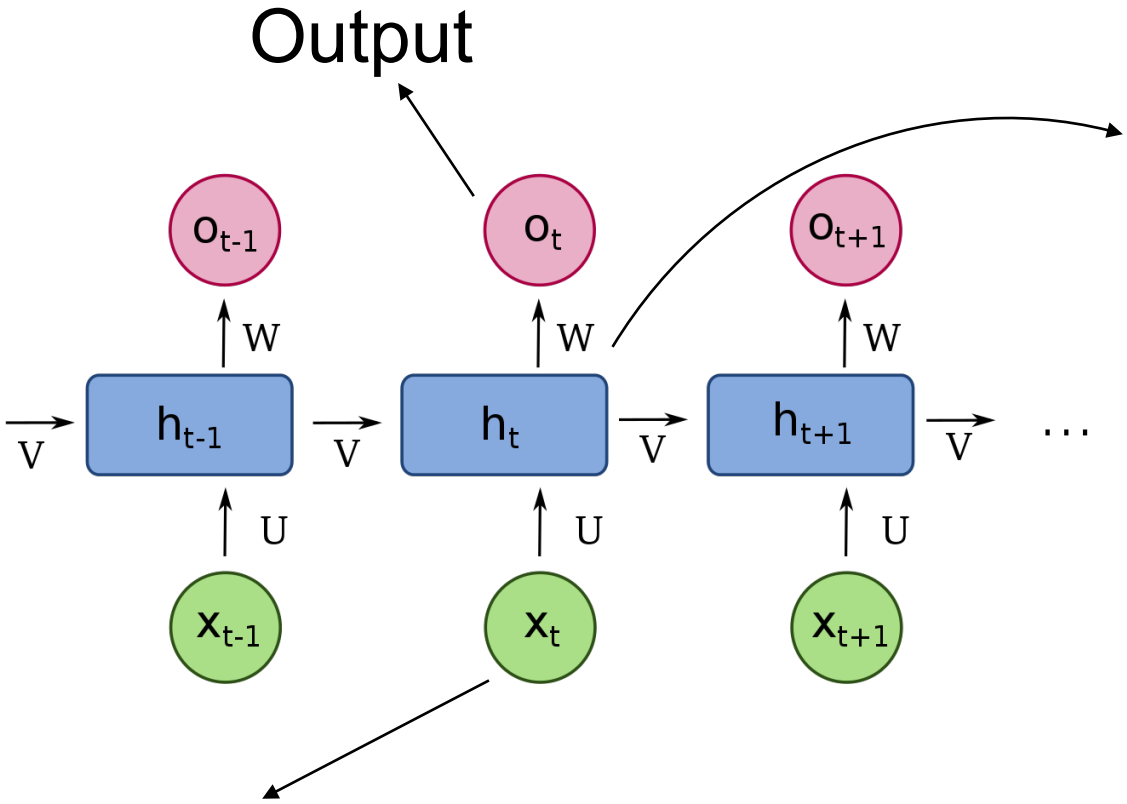
- Andre dype nevrale nettverk (e.g. Google's BERT)



Recurrent neural networks

- Moderne språkteknologiske verktøy bruker ofte **nevrale nettverk** (NN) i stedet for HMMs
- Men hovedidéene forblir de samme:
  1. Modellen beskriver mulige *transisjoner* fra tilstand til tilstand
  2. Samt hvordan observasjoner ("input") påvirker tilstanden
  3. Man bruker *decoding* til å finne den beste outputsekvensen
  4. Og modellen inneholder parametrene som *læres fra data*
- Størst forskjell: NN bruker *ikke-lineære transformasjoner* mellom vektorer i stedet for sannsynlighetsmatriser

Enkle “recurrent neural nets”



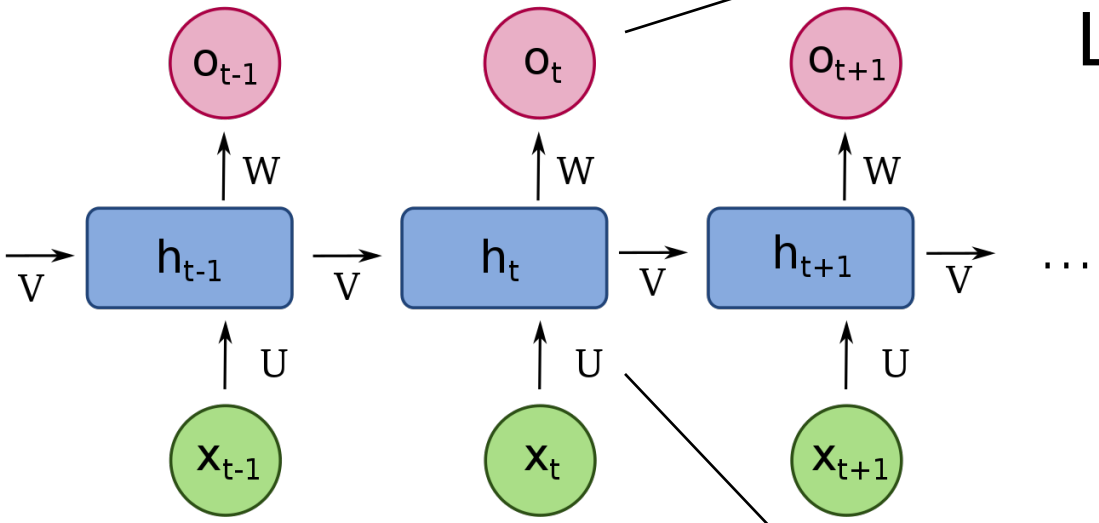
h_t er den “skjulte tilstanden” og er en mellomliggende vektor som representerer sekvensens innhold så langt

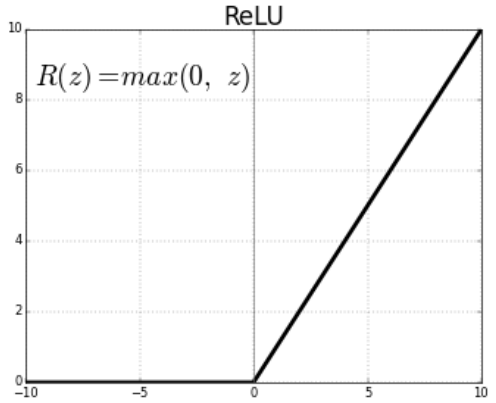
Input x_t er ofte en ordvektor (plus andre mulige trekk ved behov)

Enkle “recurrent neural nets”

$$\boldsymbol{o}_t = softmax(\boldsymbol{W}\boldsymbol{h}_t)$$

Logistisk regresjon!

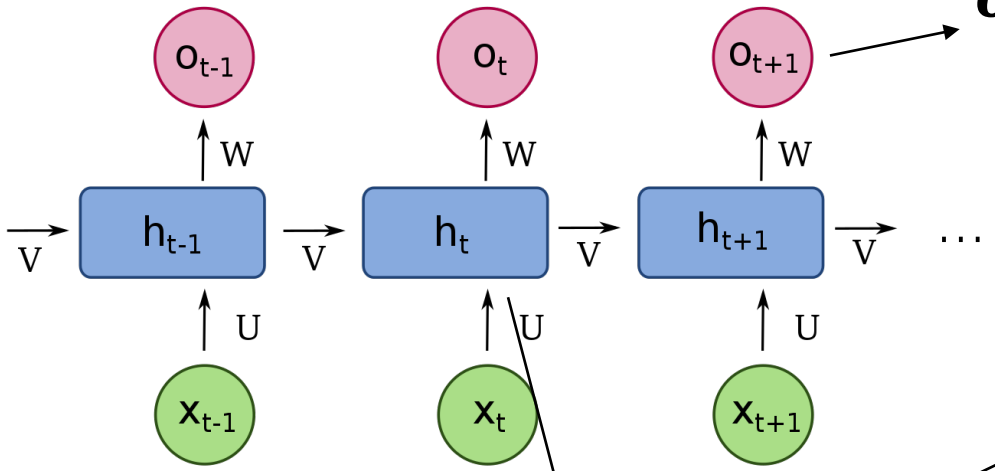




$$\boldsymbol{h}_t = \sigma(\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{V}\boldsymbol{h}_{t-1})$$

U, **V** og **W** er matriser, og σ er en ikke-lineær funksjon (f.eks. sigmoid, tanh eller ReLU)

Enkle “recurrent neural nets”



$$\boldsymbol{o}_t = softmax(\boldsymbol{W}\boldsymbol{h}_t)$$

$$\boldsymbol{W} = \begin{bmatrix} w_{11} & \dots & w_{1m} \\ \dots & \dots & \dots \\ w_{o1} & \dots & w_{om} \end{bmatrix}$$

$$\boldsymbol{U} = \begin{bmatrix} u_{11} & \dots & u_{1n} \\ \dots & \dots & \dots \\ u_{m1} & \dots & u_{mn} \end{bmatrix}$$

$$\boldsymbol{h}_t = \sigma(\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{V}\boldsymbol{h}_{t-1})$$

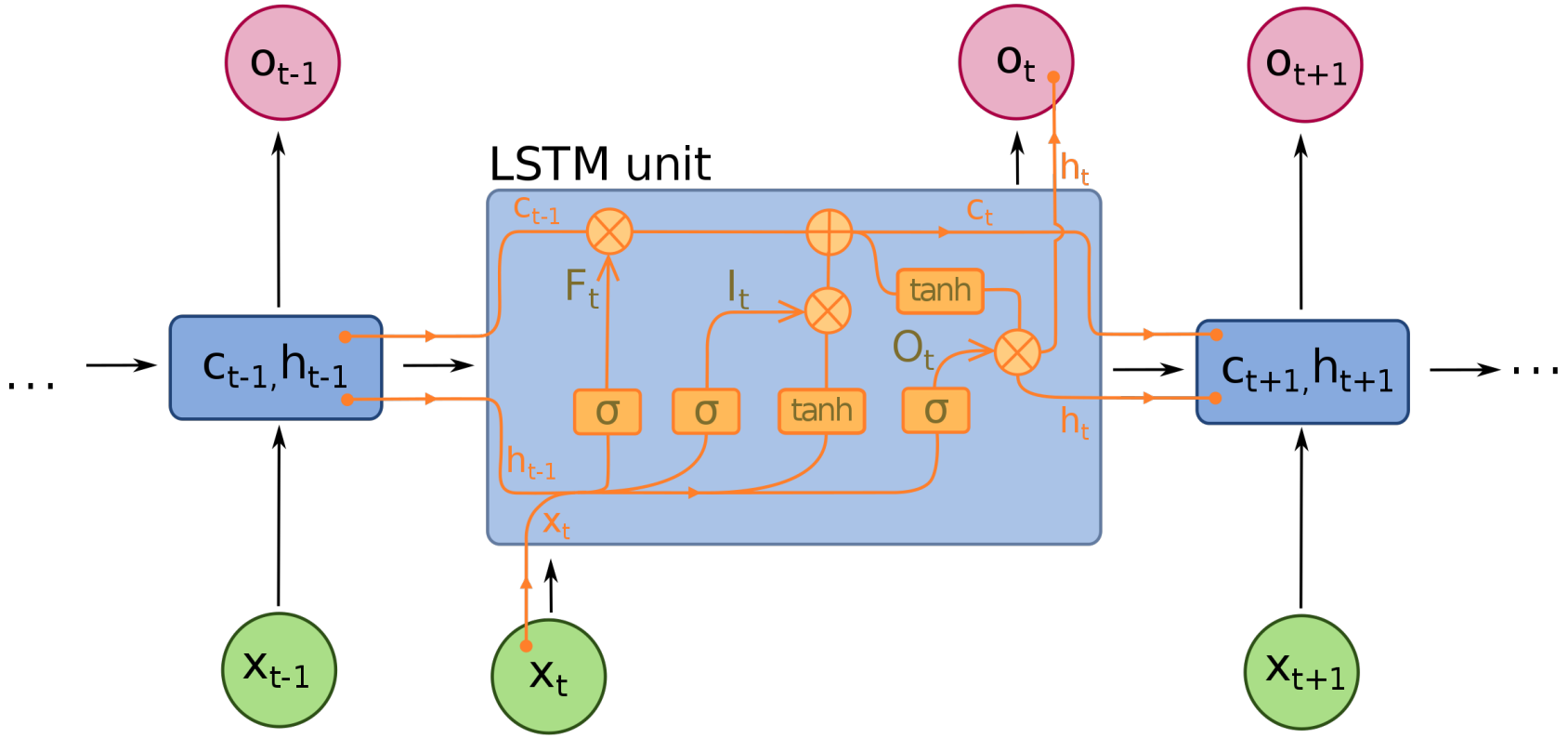
$$\boldsymbol{V} = \begin{bmatrix} v_{11} & \dots & v_{1m} \\ \dots & \dots & \dots \\ v_{m1} & \dots & v_{mm} \end{bmatrix}$$

La oss si at

- Ordvektorene $\mathbf{x}_t$ har en størrelse n=300
- Tilstandvektorene $\mathbf{h}_t$ har en størrelse m=500
- Og vi har et antall outputkategorier o=100

➔ Hvor mange parametre (lært fra data) har vi totalt?

LSTMs



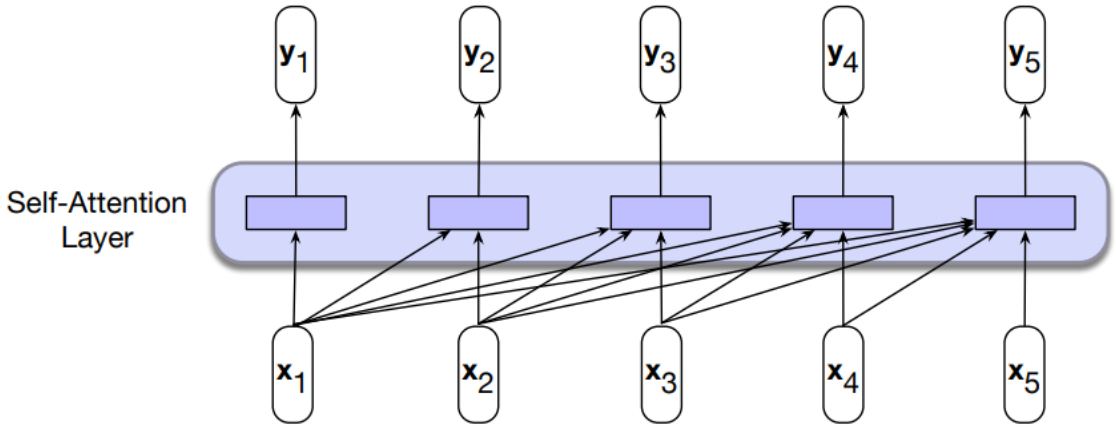
[bare til illustrasjon, dere trenger ikke å forstå hvordan LSTMs fungerer til eksamen]

Transformers

Ulemper med rekurrente nettverk:

- Representasjon av “konteksten” begrenset til tilstandsvektoren som føres gjennom sekvensen
- Treig (vanskelig å parallelisere beregniner)

Transformers tilbyr et alternativ, hvor hver ordvektoren lar seg «påvirke» av de omkringliggende ordene

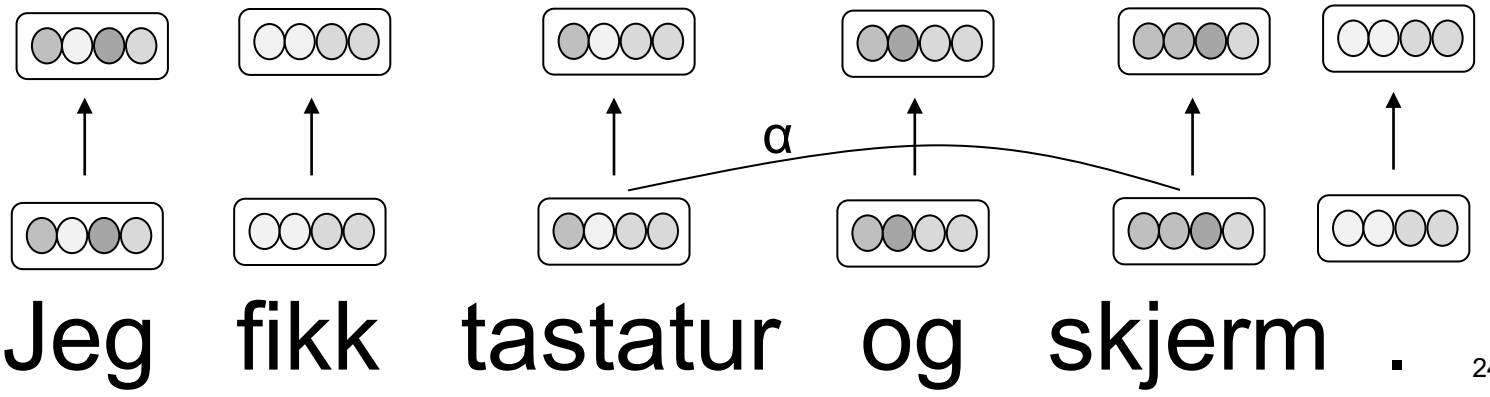


[figuren fra J&M]

Transformers

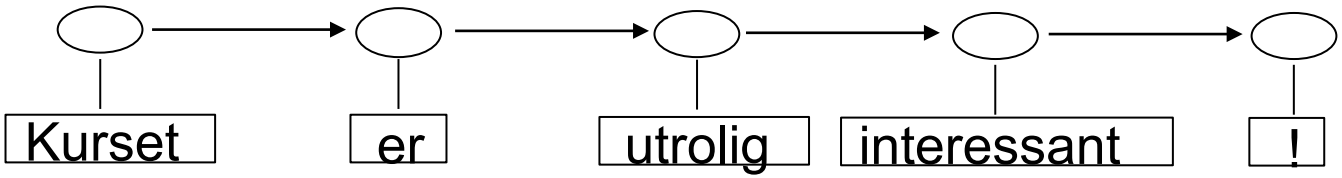
Kjernen i transformers er *self-attention:*

- For et ord som “skjerm” ser vi på konteksten, altså f.eks. ordene som kommer før (“jeg”, “fikk”, “tastatur”, “og”)
- For hvert ordpar (hovedord, kontekstord) beregner vi en *attention score* α som uttrykker hvor viktig kontekstordet (“tastatur”) er for å forstå hovedordet (“skjerm”)
- Ordvektoren er deretter oppdatert ved å ta et vektet sum av alle kontekstordene vektet med sin attention score



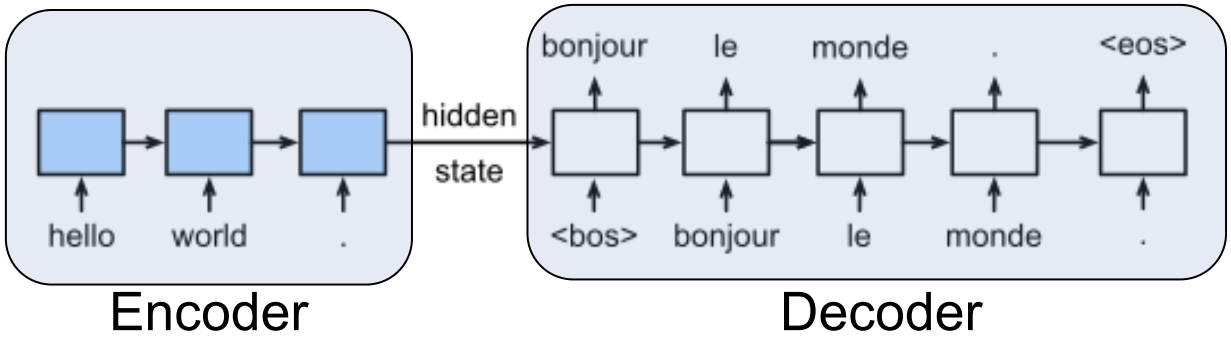
Seq2seq modeller

- Sekvensmodeller diskutert så langt har en outputsekvens av same lengde som inputsekvensen



- Hva med problemer hvor input- og outputsekvensene har ulike lengder, som i f.eks. maskinoversettelse?

  ➔ *sequence-to-sequence* (seq2seq) modeller



Oppsummering

- **Sekvensmodeller** veldig viktig i språkteknologi: språkdata tar nesten alltid en sekventiell form
- Hidden Markov Models (HMMs) er enkle men effective sekvensmodeller:
  - To deler: en *transisjonsmodell* og en *emisjonsmodell*
  - *Viterbi* brukes til å finne den beste outputsekvensen
  - Trening av HMM gjøres ved å *telle* forekomst av bestemte transisjoner og emisjoner + smoothing
- Moderne sekvensmodeller ofte basert på nevrale nettverk