# Dokument-embeddings / Markov-kjeder

Fredrik Jørgensen, Schibsted Media/UiO

# Oversikt

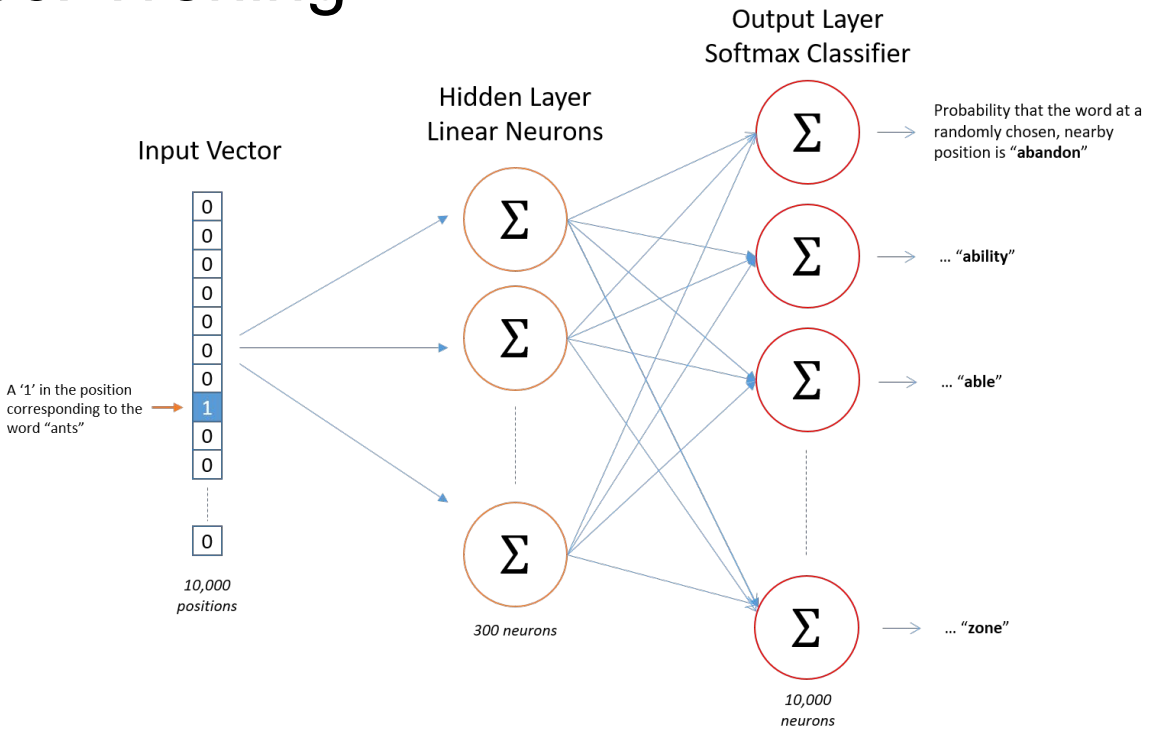
1. Dokument-embeddings: Repetisjon + ord => dokumenter
2. Markovkjeder: Vi bygger en robot-journalist!

# 1. Dokument-embeddings

# Word2Vec: Høynivå

- Trene, ikke telle
- Distribuert/kontinuerlig representasjon, ord har grad av *likhet*
- Til forskjell fra diskret representasjon, alle ord er bare *forskjellige*

# Word2vec: Trening




From https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b

# Word2vec: Likhet

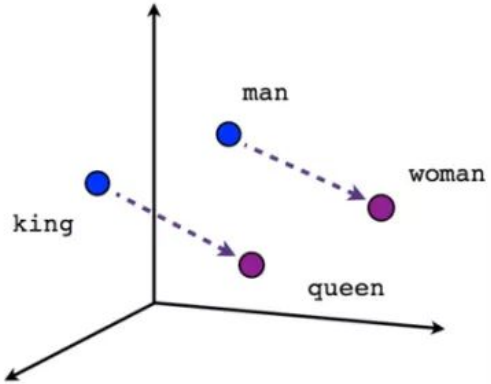
## Sweden

Most similar words

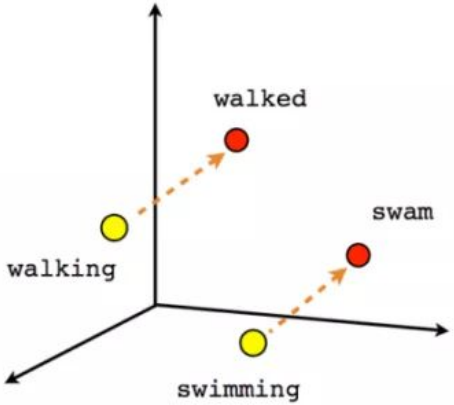
| Word | Cosine distance |
|---|---|
| norway | 0.760124 |
| denmark | 0.715460 |
| finland | 0.620022 |
| switzerland | 0.588132 |
| belgium | 0.585835 |
| netherlands | 0.574631 |
| iceland | 0.562368 |
| estonia | 0.547621 |
| slovenia | 0.531408 |
| floorball_federation | 0.529570 |
| luxembourg | 0.529477 |
| czech_republic | 0.528778 |
| slovakia | 0.526340 |
| romania | 0.524281 |
| kista | 0.522488 |
| helsinki_vantaa | 0.519936 |
| swedish | 0.519901 |
| balrog_ik | 0.514556 |
| portugal | 0.502495 |
| russia | 0.500196 |
| slovakia_slovenia | 0.496051 |
| ukraine | 0.495712 |

From https://www.slideshare.net/hen_drik/hacking-human-language-pydata-london

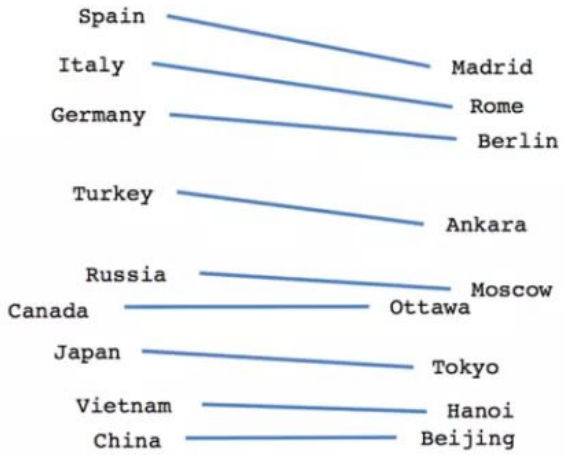
# Word2Vec: Egenskaper




Male-Female




Verb tense




Country-Capital

From https://www.tensorflow.org/tutorials/representation/word2vec

# Word2Vec: Selve modellen

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 290 | 291 | 292 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **fox** | -0.348680 | -0.077720 | 0.177750 | -0.094953 | -0.452890 | 0.237790 | 0.209440 | 0.037886 | 0.035064 | 0.899010 | ... | -0.283050 | 0.270240 | -0.654800 | 0.10 |
| **ham** | -0.773320 | -0.282540 | 0.580760 | 0.841480 | 0.258540 | 0.585210 | -0.021890 | -0.463680 | 0.139070 | 0.658720 | ... | 0.464470 | 0.481400 | -0.829200 | 0.35 |
| **brown** | -0.374120 | -0.076264 | 0.109260 | 0.186620 | 0.029943 | 0.182700 | -0.631980 | 0.133060 | -0.128980 | 0.603430 | ... | -0.015404 | 0.392890 | -0.034826 | -0.72 |
| **beautiful** | 0.171200 | 0.534390 | -0.348540 | -0.097234 | 0.101800 | -0.170860 | 0.295650 | -0.041816 | -0.516550 | 2.117200 | ... | -0.285540 | 0.104670 | 0.126310 | 0.12 |
| **jumps** | -0.334840 | 0.215990 | -0.350440 | -0.260020 | 0.411070 | 0.154010 | -0.386110 | 0.206380 | 0.386700 | 1.460500 | ... | -0.107030 | -0.279480 | -0.186200 | -0.54 |
| **eggs** | -0.417810 | -0.035192 | -0.126150 | -0.215930 | -0.669740 | 0.513250 | -0.797090 | -0.068611 | 0.634660 | 1.256300 | ... | -0.232860 | -0.139740 | -0.681080 | -0.37 |
| **beans** | -0.423290 | -0.264500 | 0.200870 | 0.082187 | 0.066944 | 1.027600 | -0.989140 | -0.259950 | 0.145960 | 0.766450 | ... | 0.048760 | 0.351680 | -0.786260 | -0.36 |
| **sky** | 0.312550 | -0.303080 | 0.019587 | -0.354940 | 0.100180 | -0.141530 | -0.514270 | 0.886110 | -0.530540 | 1.556600 | ... | -0.667050 | 0.279110 | 0.500970 | -0.27 |
| **bacon** | -0.430730 | -0.016025 | 0.484620 | 0.101390 | -0.299200 | 0.761820 | -0.353130 | -0.325290 | 0.156730 | 0.873210 | ... | 0.304240 | 0.413440 | -0.540730 | -0.03 |
| **breakfast** | 0.073378 | 0.227670 | 0.208420 | -0.456790 | -0.078219 | 0.601960 | -0.024494 | -0.467980 | 0.054627 | 2.283700 | ... | 0.647710 | 0.373820 | 0.019931 | -0.03 |
| **toast** | 0.130740 | -0.193730 | 0.253270 | 0.090102 | -0.272580 | -0.030571 | 0.096945 | -0.115060 | 0.484000 | 0.848380 | ... | 0.142080 | 0.481910 | 0.045167 | 0.05 |
| **today** | -0.156570 | 0.594890 | -0.031445 | -0.077586 | 0.278630 | -0.509210 | -0.066350 | -0.081890 | -0.047986 | 2.803600 | ... | -0.326580 | -0.413380 | 0.367910 | -0.26 |
| **blue** | 0.129450 | 0.036518 | 0.032298 | -0.060034 | 0.399840 | -0.103020 | -0.507880 | 0.076630 | -0.422920 | 0.815730 | ... | -0.501280 | 0.169010 | 0.548250 | -0.31 |
| **green** | -0.072368 | 0.233200 | 0.137260 | -0.156630 | 0.248440 | 0.349870 | -0.241700 | -0.091426 | -0.530150 | 1.341300 | ... | -0.405170 | 0.243570 | 0.437300 | -0.46 |
| **kings** | 0.259230 | -0.854690 | 0.360010 | -0.642000 | 0.568530 | -0.321420 | 0.173250 | 0.133030 | -0.089720 | 1.528600 | ... | -0.470090 | 0.063743 | -0.545210 | -0.19 |
| **dog** | -0.057120 | 0.052685 | 0.003026 | -0.048517 | 0.007043 | 0.041856 | -0.024704 | -0.039783 | 0.009614 | 0.308416 | ... | 0.003257 | -0.036864 | -0.043878 | 0.00 |
| **sausages** | -0.174290 | -0.064869 | -0.046976 | 0.287420 | -0.128150 | 0.647630 | 0.056315 | -0.240440 | -0.025094 | 0.502220 | ... | 0.302240 | 0.195470 | -0.653980 | -0.29 |
| **lazy** | -0.353320 | -0.299710 | -0.176230 | -0.321940 | -0.385640 | 0.586110 | 0.411160 | -0.418680 | 0.073093 | 1.486500 | ... | 0.402310 | -0.038554 | -0.288670 | -0.24 |
| **love** | 0.139490 | 0.534530 | -0.252470 | -0.125650 | 0.048748 | 0.152440 | 0.199060 | -0.065970 | 0.128830 | 2.055900 | ... | -0.124380 | 0.178440 | -0.099469 | 0.00 |
| **quick** | -0.445630 | 0.191510 | -0.249210 | 0.465900 | 0.161950 | 0.212780 | -0.046480 | 0.021170 | 0.417660 | 1.686900 | ... | -0.329460 | 0.421860 | -0.039543 | 0.15 |

20 rows × 300 columns

# Fra ord til dokumenter

- Word embeddings gir oss en god semantisk representasjon av *ord*
- Vi kan finne nærmeste ord i et vektorrom
- Kan vi finne nærmeste *setning* eller *dokument* i et vektorrom?
- word embeddings => document embeddings

# Dokument-embeddings1 : Gjennomsnitt

Eksempel:

- Setningen **“vektorer er moro!”**
- En word2vec-modell

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **vektorer** | 0.374540 | 0.950714 | 0.731994 | 0.598658 | 0.156019 | 0.155995 | 0.058084 | 0.866176 | 0.601115 | 0.708073 |
| **er** | 0.020584 | 0.969910 | 0.832443 | 0.212339 | 0.181825 | 0.183405 | 0.304242 | 0.524756 | 0.431945 | 0.291229 |
| **moro** | 0.611853 | 0.139494 | 0.292145 | 0.366362 | 0.456070 | 0.785176 | 0.199674 | 0.514234 | 0.592415 | 0.046450 |
| **!** | 0.607545 | 0.170524 | 0.065052 | 0.948886 | 0.965632 | 0.808397 | 0.304614 | 0.097672 | 0.684233 | 0.440152 |

# Dokument-embeddings 1: Gjennomsnitt

Eksempel:

- Setningen **“vektorer er moro!”**
- En word2vec-modell

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **vektorer** | 0.374540 | 0.950714 | 0.731994 | 0.598658 | 0.156019 | 0.155995 | 0.058084 | 0.866176 | 0.601115 | 0.708073 |
| **er** | 0.020584 | 0.969910 | 0.832443 | 0.212339 | 0.181825 | 0.183405 | 0.304242 | 0.524756 | 0.431945 | 0.291229 |
| **moro** | 0.611853 | 0.139494 | 0.292145 | 0.366362 | 0.456070 | 0.785176 | 0.199674 | 0.514234 | 0.592415 | 0.046450 |
| **!** | 0.607545 | 0.170524 | 0.065052 | 0.948886 | 0.965632 | 0.808397 | 0.304614 | 0.097672 | 0.684233 | 0.440152 |
| **μ("vektorer er moro !")** | 0.403631 | 0.557661 | 0.480408 | 0.531561 | 0.439886 | 0.483243 | 0.216653 | 0.500710 | 0.577427 | 0.371476 |

# Dokument-embeddings 2: Gjennomsnitt med TFIDF

Eksempel:

- Setningen **“vektorer er moro!”**
- En word2vec-modell
- En (TF)IDF-vektor

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

| | idf |
|---|---|
| **vektorer** | 6.907755 |
| **er** | 0.693147 |
| **moro** | 4.605170 |
| **!** | 2.302585 |

# Dokument-embeddings 2: Gjennomsnitt med TFIDF

Eksempel:

- Setningen **“vektorer er moro!”**
- En word2vec-modell
- En (TF)IDF-vektor
- Vekte embeddingene med (TF)IDF:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **idf('vektorer')** | 2.587231 | 6.567302 | 5.056435 | 4.135386 | 1.077739 | 1.077572 | 0.401227 | 5.983333 | 4.152355 | 4.891192 |
| **idf('er')** | 0.014268 | 0.672290 | 0.577005 | 0.147182 | 0.126031 | 0.127126 | 0.210885 | 0.363733 | 0.299401 | 0.201865 |
| **idf('moro')** | 2.817687 | 0.642393 | 1.345376 | 1.687159 | 2.100280 | 3.615869 | 0.919532 | 2.368137 | 2.728170 | 0.213912 |
| **idf('!')** | 1.398924 | 0.392646 | 0.149787 | 2.184890 | 2.223450 | 1.861404 | 0.701399 | 0.224898 | 1.575505 | 1.013489 |
| **μ(idf("vektorer er moro !"))** | 1.704527 | 2.068658 | 1.782151 | 2.038654 | 1.381875 | 1.670493 | 0.558261 | 2.235025 | 2.188858 | 1.580114 |

# (Dokument-embeddings 3: TFIDF + PCA)

- Kort om PCA: Matrise-faktorisering, finner de dimensjonene som har størst varians

  For spesielt interesserte: A Simple But Tough-to-beat Baseline For Sentence Embeddings

- Hypotese: Ord har mest varians i den syntaktiske dimensjonen
- Fjerne første PCA komponente(ne), dvs fjerne varians, dvs “fjerne syntaks”
- Vekte med TFIDF

# Dokument-embeddings basert på word embeddings

+

- Enkelt å lage
- Ingen trening utover word2vec
- Relativt enkelt å forstå
- Distribuert/kontinuerlig representasjon, *ord* har likhet (jfr. BOW, diskret modell)
  - The pizza was great
  - The margherita was awesome
  - The dog was sick

# Dokument-embeddings basert på word embeddings

-

- Fungerer dårlig på lengre tekster. Best på setninger.
- “Regression to the mean”

# Farvel til ord og embeddings

# Hei hei Andrey Markov

# 2. Sekvens-modellering og Markov-kjeder

# Bokmålsordboka

**Oppslagsord** **Ordbokartikkel**

sekvens

**sekvens** m1 (fra middelalderlatin, av latin *sequi* 'følge')

**1** i musikk motiv som gjentas i forandret toneleie

**2** hymne som blir brukt i katolsk messe

**3** rekke av elementer som danner et hele, for eksempel spillkort av samme farge, filmopptak og lignende
*vise noen sekvenser fra filmen*

# Sekvenser

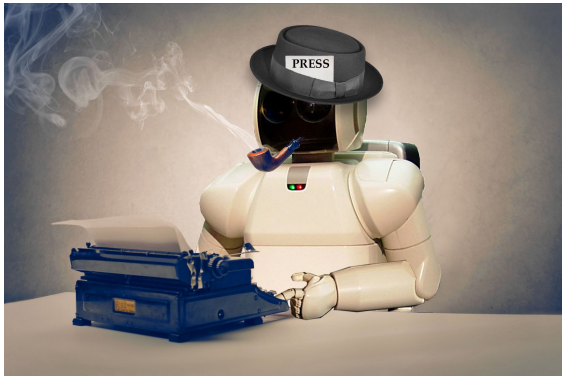
**Sekvens = Elementer med en ordning/rekkefølge**

Språk er ofte sekvenser. Eksempler:

- Ord i rekkefølge danner fraser/setninger
- Setninger i rekkefølge danner en tekst

# Markov-kjeder

- Vi lager en robot-journalist!
- Ved hjelp av en sekvens-modell som heter Markov-kjeder (Markov Chains)
- Roboten skal få lære av tekst skrevet av ekte journalister
- Og så begynne å skrive selv…
- Men først: Markov-kjeder



# Markov-kjeder

“Markovkjede er i statistikkfaget en serie med tilfeldige variabler som er egnet til å beskrive prosesser hvor den fremtidige tilstanden kun er avhengig av hva den er nå og ikke hva den har vært.” [1]

[1] https://no.wikipedia.org/wiki/Markovkjede (16/4-2018)

# Markov-kjeder

“Markovkjede er i statistikkfaget en **serie** med tilfeldige variabler som er egnet til å beskrive prosesser hvor den fremtidige tilstanden kun er avhengig av hva den er nå og ikke hva den har vært.”

- *En serie eller sekvens, akkurat som tekst*

# Markov-kjeder

“Markovkjede er i statistikkfaget en serie med **tilfeldige** variabler som er egnet til å beskrive prosesser hvor den fremtidige tilstanden kun er avhengig av hva den er nå og ikke hva den har vært.”

- En serie eller sekvens, akkurat som tekst
- *Vi trenger sannsynligheter*

# Markov-kjeder

“Markovkjede er i statistikkfaget en serie med tilfeldige variabler som er egnet til å **beskrive prosesser** hvor den fremtidige tilstanden kun er avhengig av hva den er nå og ikke hva den har vært.”

- En serie eller sekvens, akkurat som tekst
- Vi trenger sannsynligheter
- *Her skal vi lage en modell*

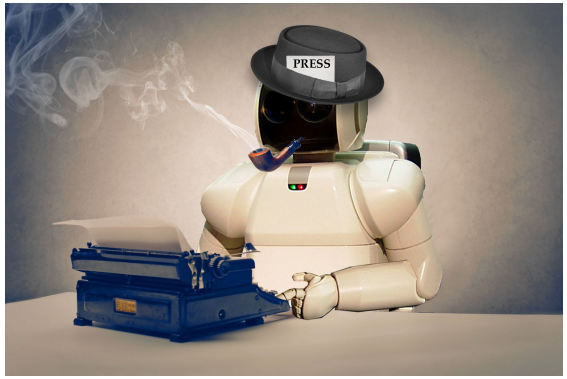
# Markov-kjeder

“Markovkjede er i statistikkfaget en serie med tilfeldige variabler som er egnet til å beskrive prosesser **hvor den fremtidige tilstanden kun er avhengig av hva den er nå og ikke hva den har vært**.”

- En serie eller sekvens, akkurat som tekst
- Vi trenger sannsynligheter
- Her skal vi lage en modell
- *Dette kalles “Markov-egenskapen” og betyr at vi ikke trenger å huske tidligere tilstander, kun den nåværende*

# Markov-kjeder

- Dette skal roboten lære:
- **Input:** Masse overskrifter
- **Output:** Overskrifter som ligner på det roboten har sett
- Overskrifter er sekvenser av ord



# Markov-kjeder

“Markovkjede er i statistikkfaget en serie med tilfeldige variabler som er egnet til å beskrive prosesser hvor den fremtidige tilstanden **kun er avhengig av hva den er nå og ikke hva den har vært**.”

$$\Pr( X_{n+1} = x \mid X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n )$$

# Markov-kjeder

“Markovkjede er i statistikkfaget en serie med tilfeldige variabler som er egnet til å beskrive prosesser hvor den fremtidige tilstanden **kun er avhengig av hva den er nå og ikke hva den har vært**.”

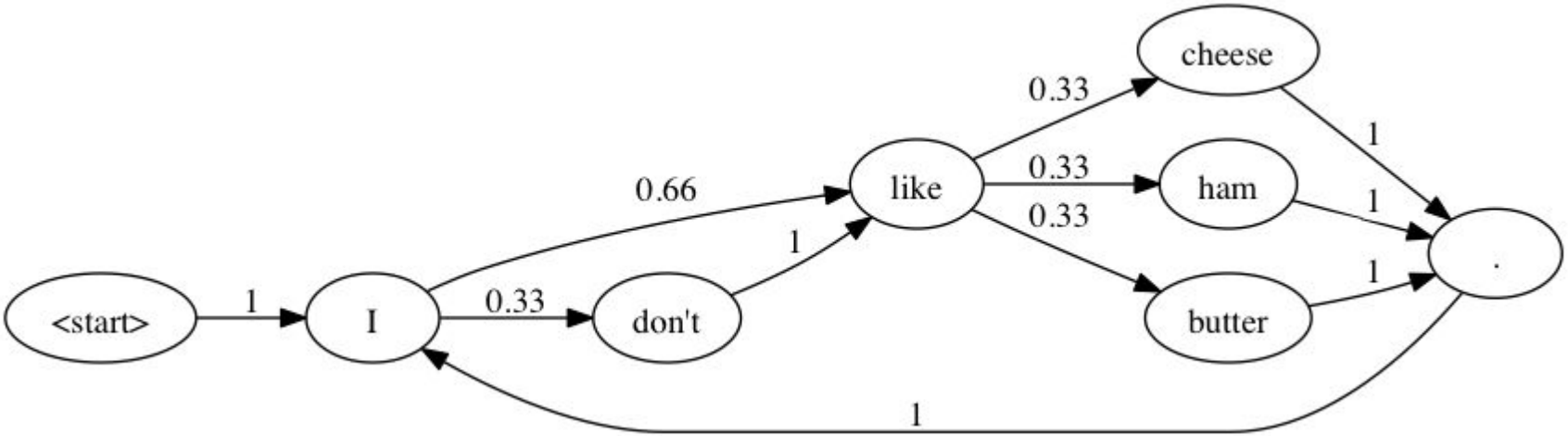
$$\Pr( X_{n+1} = x \mid X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n )$$

$$=$$

$$\Pr( X_{n+1} = x \mid X_n = x_n )$$

**(Markov assumption)**

# Markov-kjeder: Grafisk representasjon



# Markov-kjeder: Definisjon i J&M

Fra Jurafsky & Martin (3rd ed.), kap. A.1:

| | |
|---|---|
| $Q = q_1 q_2 \ldots q_N$ | a set of $N$ **states** |
| $A = a_{01} a_{02} \ldots a_{n1} \ldots a_{nn}$ | a **transition probability matrix** $A$, each $a_{ij}$ representing the probability of moving from state $i$ to state $j$, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$ |
| $q_0, q_F$ | a special **start state** and **end (final) state** that are not associated with observations |

# Markov-kjeder

“Markovkjede er i statistikkfaget en serie med tilfeldige variabler som er egnet til å beskrive prosesser hvor den fremtidige tilstanden kun er avhengig av hva den er nå og ikke hva den har vært.”

***Kan vi bygge det? Klart vi kan!***

# 3. Implementasjon

# Markov-kjeder

```
class MarkovChain:

    def __init__(self):
        self.memory = defaultdict(lambda: [])
```

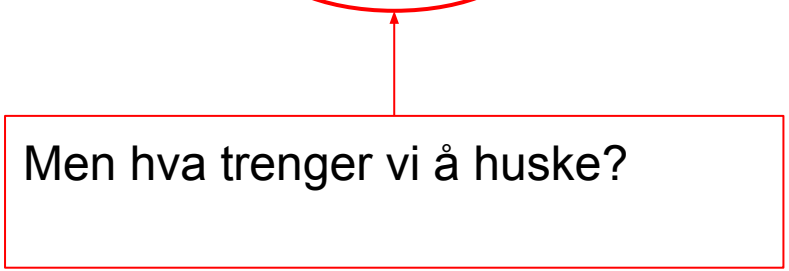
# Markov-kjeder

```
class MarkovChain:

    def __init__(self):
        self.memory = defaultdict(lambda: [])
```

For å lære fra data, må vi huske litt...

# Markov-kjeder

```
class MarkovChain:

    def __init__(self):
        self.memory = defaultdict(lambda: [])
```



# Markov-kjeder

```
class MarkovChain:

    def __init__(self):
        self.memory = defaultdict(lambda: [])
```

Vi må huske *forrige tilstand*
og *nåværende tilstand*

# Markov-kjeder

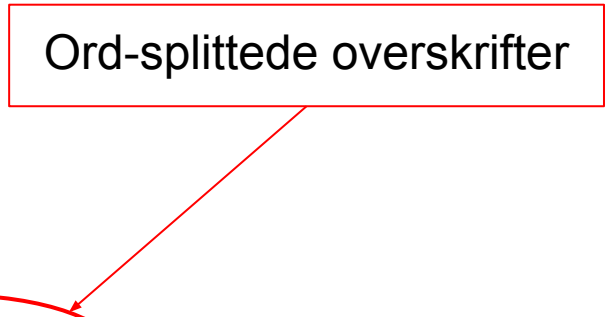
```
class MarkovChain:

    def __init__(self):
        self.memory = defaultdict(lambda: [])
```

**Tilstand = ord.** Vi må huske *forrige ord* og *nåværende ord*

# Markov-kjeder

```
class MarkovChain:

    def fit(self, sentences):
        for sentence in sentences:
            tokens = sentence.split(" ")
            bigrams = [(tokens[i], tokens[i + 1])
                for i in range(0, len(tokens) - 1)]
            for (state, output) in bigrams:
                self.memory[state].append(output)
```

# Markov-kjeder




```
class MarkovChain:

    def fit(self, sentences):
        for sentence in sentences:
            tokens = sentence.split(" ")
            bigrams = [(tokens[i], tokens[i + 1])
                for i in range(0, len(tokens) - 1)]
            for (state, output) in bigrams:
                self.memory[state].append(output)
```

# Markov-kjeder

```
class MarkovChain:

    def fit(self, sentences):
        for sentence in sentences:
            tokens = sentence.split(" ")
            bigrams = [(tokens[i], tokens[i + 1])
                for i in range(0, len(tokens) - 1)]
            for (state, output) in bigrams:
                self.memory[state].append(output)
```

Par av *forrige ord* og *nåværende ord*

# Markov-kjeder

```
class MarkovChain:

    def fit(self, sentences):
        for sentence in sentences:
            tokens = sentence.split(" ")
            bigrams = [(tokens[i], tokens[i + 1])
                for i in range(0, len(tokens) - 1)]
            for (state, output) in bigrams:
                self.memory[state].append(output)
```

Lage i ‘memory´:
En dict fra *forrige ord* til
en liste av *nåværende ord*

# Markov-kjeder

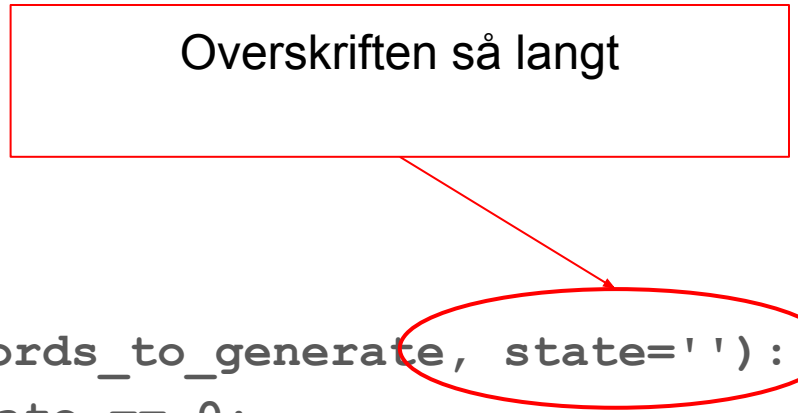
```
class MarkovChain:

    def generate(self, words_to_generate, state=''):
        if words_to_generate == 0:
            return state

        next_word = self._next(state)
        return state + ' ' +
            self.generate(words_to_generate - 1, next_word)
```

# Markov-kjeder

Lengen på overskriften
Vi skal lage

```
class MarkovChain:

    def generate(self, words_to_generate, state=''):
        if words_to_generate == 0:
            return state

        next_word = self._next(state)
        return state + ' ' +
            self.generate(words_to_generate - 1, next_word)
```

# Markov-kjeder



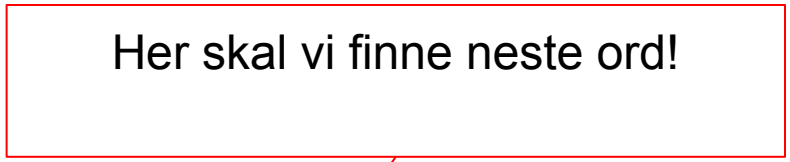

```
class MarkovChain:

    def generate(self, words_to_generate, state=''):
        if words_to_generate == 0:
            return state

        next_word = self._next(state)
        return state + ' ' +
            self.generate(words_to_generate - 1, next_word)
```

# Markov-kjeder

Dette er en *rekursiv* funksjon, dvs. at den kaller seg selv

```
class MarkovChain:

    def generate(self, words_to_generate, state=''):
        if words_to_generate == 0:
            return state

        next_word = self._next(state)
        return state + ' ' +
            self.generate(words_to_generate - 1, next_word)
```

# Markov-kjeder




```
class MarkovChain:

    def generate(self, words_to_generate, state=''):
        if words_to_generate == 0:
            return state

        next_word = self._next(state)
        return state + ' ' +
            self.generate(words_to_generate - 1, next_word)
```

# Markov-kjeder

```
class MarkovChain:

    def _next(self, current_state):
        next_possible = self.memory.get(current_state)
        if not next_possible:
            next_possible = self.memory.keys()
        return random.sample(next_possible, 1)[0]
```

# Markov-kjeder

Generere neste ord/tilstand fra nåværende ord/tilstand

```
class MarkovChain:

    def _next(self, current_state):
        next_possible = self.memory.get(current_state)
        if not next_possible:
            next_possible = self.memory.keys()
        return random.sample(next_possible, 1)[0]
```

# Markov-kjeder

Quiz: Hva slags datastruktur er dette?

```
class MarkovChain:

    def _next(self, current_state):
        next_possible = self.memory.get(current_state)
        if not next_possible:
            next_possible = self.memory.keys()
        return random.sample(next_possible, 1)[0]
```

# Markov-kjeder

Fasit: En liste med alle ord/tilstander vi har sett som følger 'current_state'

```
class MarkovChain:

    def _next(self, current_state):
        next_possible = self.memory.get(current_state)
        if not next_possible:
            next_possible = self.memory.keys()
        return random.sample(next_possible, 1)[0]
```

# Markov-kjeder

Hvis vi ikke har sett dette ordet, tar vi et tilfeldig ord...

```
class MarkovChain:

    def _next(self, current_state):
        next_possible = self.memory.get(current_state)
        if not next_possible:
            next_possible = self.memory.keys()
        return random.sample(next_possible, 1)[0]
```

# Markov-kjeder

Vi velger et tilfeldig neste ord

```
class MarkovChain:

    def _next(self, current_state):
        next_possible = self.memory.get(current_state)
        if not next_possible:
            next_possible = self.memory.keys()
        return random.sample(next_possible, 1)[0]
```

# Markov-kjeder

```
class MarkovChain:
    def __init__(self):
        self.memory = defaultdict(lambda: [])

    def fit(self, sentences):
        for sentence in sentences:
            tokens = sentence.split(' ')
            bigrams = [(tokens[i], tokens[i + 1]) for i in range(0, len(tokens) - 1)]
            for (key, value) in bigrams:
                self.memory[key].append(value)

    def generate(self, sentence_length, state=''):
        if sentence_length == 0:
            return state
        next_word = self._next(state)
        return state + ' ' + self.generate(sentence_length - 1, next_word)

    def _next(self, current_state):
        next_possible = self.memory.get(current_state)
        if not next_possible:
            next_possible = self.memory.keys()
        return random.sample(next_possible, 1)[0]
```

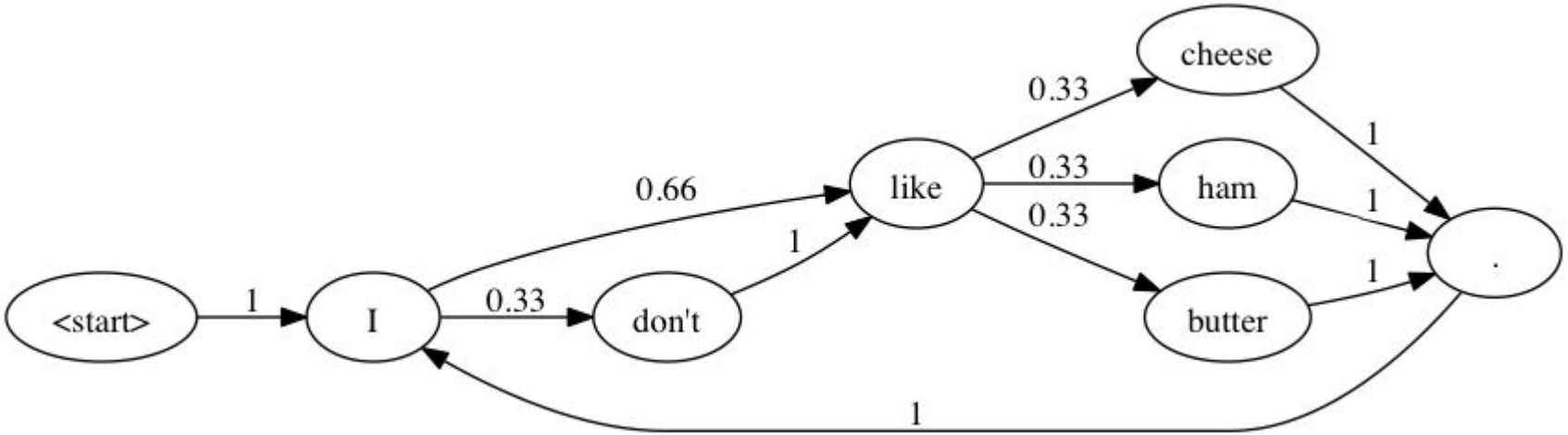
Mindre enn
30 linjer
Med kode!

# Markov-kjeder

Demo

# Markov-kjeder

Grafisk representasjon:



# Demo (v.1)

# 4. Forbedringer

# Forbedringer

Hmm, det er en start, men funker ikke helt bra…

- Det hadde vært fint om modellen selv visste når en setning var ferdig
- Modellen husker ikke så langt tilbake, bare ett ord
- Det er mye redundant data i modellen

# Forbedringer

Hmm, det er en start, men funker ikke helt bra…

- **Det hadde vært fint om modellen selv visste når en setning var ferdig**
  - **Vi introduserer setningsgrenser: spesielle start- og slutt-ord**
  - **F eks <SOS> og <EOS>**
  - **Vi legger til disse når vi trener modellen**
- Modellen husker ikke så langt tilbake, bare ett ord
- Det er mye redundant data i modellen

# Forbedringer

Fra Jurafsky & Martin (3rd ed.), kap. A.1:

| | |
|---|---|
| $Q = q_1 q_2 \ldots q_N$ | a set of $N$ **states** |
| $A = a_{01} a_{02} \ldots a_{n1} \ldots a_{nn}$ | a **transition probability matrix** $A$, each $a_{ij}$ representing the probability of moving from state $i$ to state $j$, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$ |
| $q_0, q_F$ | a special **start state** and **end (final) state** that are not associated with observations |

# Forbedringer

Fra Jurafsky & Martin (3rd ed.), kap. A.1:

| | |
|---|---|
| $Q = q_1 q_2 \ldots q_N$ | a set of $N$ **states** |
| $A = a_{01} a_{02} \ldots a_{n1} \ldots a_{nn}$ | a **transition probability matrix** $A$, each $a_{ij}$ representing the probability of moving from state $i$ to state $j$, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$ |
| $q_0, q_F$ | a special **start state** and **end (final) state** that are not associated with observations |

# Demo (v.2)

# Forbedringer

Hmm, det er en start, men funker ikke helt bra…

- Det hadde vært fint om modellen selv visste når en setning var ferdig
- **Modellen husker ikke så langt tilbake, bare ett ord**
  - **Forrige tilstand kan være mer enn bare *ett* ord**
- Det er mye redundant data i modellen

# Demo (v.3)

(Kan slutte her hvis klokka er 12)

# Forbedringer

Hmm, det er en start, men funker ikke helt bra…

- Det hadde vært fint om modellen selv visste når en setning var ferdig
- Modellen husker ikke så langt tilbake, bare ett ord
- **Det er mye redundant data i modellen**
  - **I stedet for å lagre alle forekomster etter ord *X* i en liste, kan vi lagre sannsynlighetene for hvert ord**
  - **Nå: ‘i’ => [‘dag’, ‘morgen’, ‘dag’, ‘dag’, ‘dag’, ‘morgen’, ...]**
  - **Bedre: ‘i’ => {‘dag’: 0.2, ‘morgen’: 0.1, …}**

# Forbedringer

Hmm, det er en start, men funker ikke helt bra…

- Det hadde vært fint om modellen selv visste når en setning var ferdig
- Modellen husker ikke så langt tilbake, bare ett ord
- **Det er mye redundant data i modellen**
  - **I stedet for å lagre alle forekomster etter ord *X* i en liste, kan vi lagre sannsynlighetene som en matrise**
  - **Nå: 'i' => ['dag', 'morgen', 'dag', 'dag', 'dag', 'morgen', ...]**
  - **Bedre: 'i' => {'dag': 0.2, 'morgen': 0.1, …}**
  - **Best: (Markov) matrise**

| | i | dag | morgen |
|---|---|---|---|
| i | 0 | 0.2 | 0.1 |
| dag | 0.00003 | | |
| morgen | 0 | 0 | 0 |

# Markov-kjeder

Fra Jurafsky & Martin (3rd ed.), kap. A.1:

| | |
|---|---|
| $Q = q_1 q_2 \ldots q_N$ | a set of $N$ **states** |
| $A = a_{01} a_{02} \ldots a_{n1} \ldots a_{nn}$ | a **transition probability matrix** $A$, each $a_{ij}$ representing the probability of moving from state $i$ to state $j$, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$ |
| $q_0, q_F$ | a special **start state** and **end (final) state** that are not associated with observations |