Statistical language model (SLM)

- Content today:
  - SLM methods
  - SLM applications
- Presented by *Mikko Kurimo*
- Pics from *Sami Virpioja, Kalle Palomäki, Bryan Pellom, Steve Renals, Dan Jurafsky and Tomas Mikolov – thanks!*

Goals of today

1. Learn how to model language by statistical methods
2. Learn basic idea of neural language modeling
3. Know some typical SLM **methods and applications**

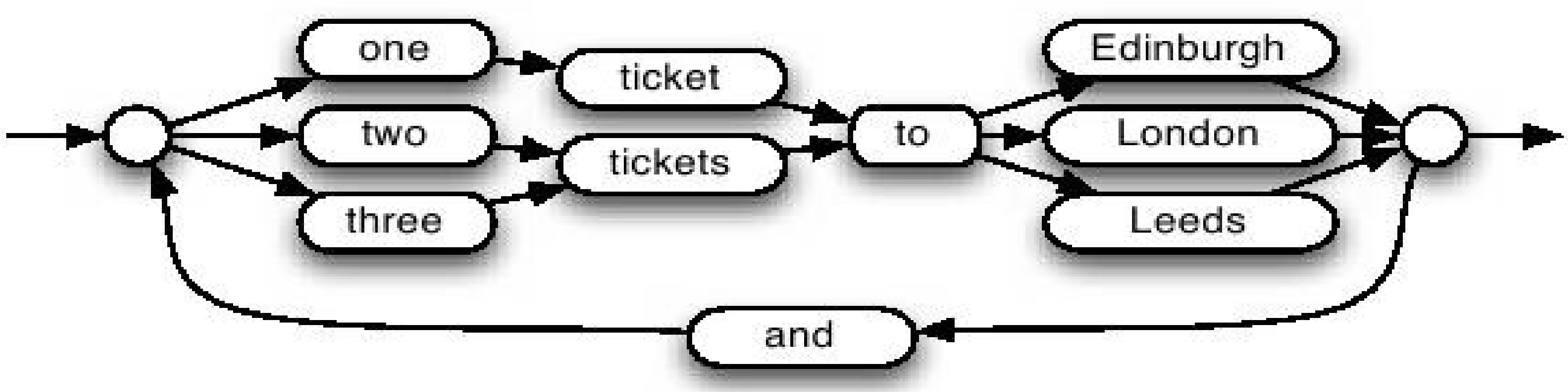
Statistical Language Model

- Model of a *natural language* that predicts the *probability distribution* of words and sentences in a text
- Often used to determine which is the most probable word or sentence in given conditions or context
- Estimated by counting word frequencies and dependencies in large text corpora
- Has to deal with: *big data, noisy data, sparse data, computational efficiency*

Some historical landmarks of SLMs

- Markov chains (Markov, 1913)
- N-grams (Shannon, 1948)
- Predicting unseen events (Good, 1953)
- Landmarks at *Aalto University* (Helsinki Univ. of Technology)
  - Dynamically expanding context (Kohonen, 1986)
  - Self-organizing semantic maps (Ritter and Kohonen, 1989)
  - WEBSOM for organizing text collections (Kohonen, 1996)
  - Morfessor for unsupervised analysis of words (Lagus. 2002)
  - Varigram LM for sequencies of words (Siivola, 2005)
  - Unlimited vocabulary LMs for speech recognition (Hirsimäki, 2006)
  - Class n-gram models for very large vocabulary speech recognition of Finnish and Estonian (Varjokallio, 2016)
  - An Extensible Toolkit for Neural Network LMs (Enarvi, 2016)

A simple statistical language model



- Limited domain models, constructed by hand
- Transition probabilities can be estimated statistically
- Only a very limited set of sentences are recognized

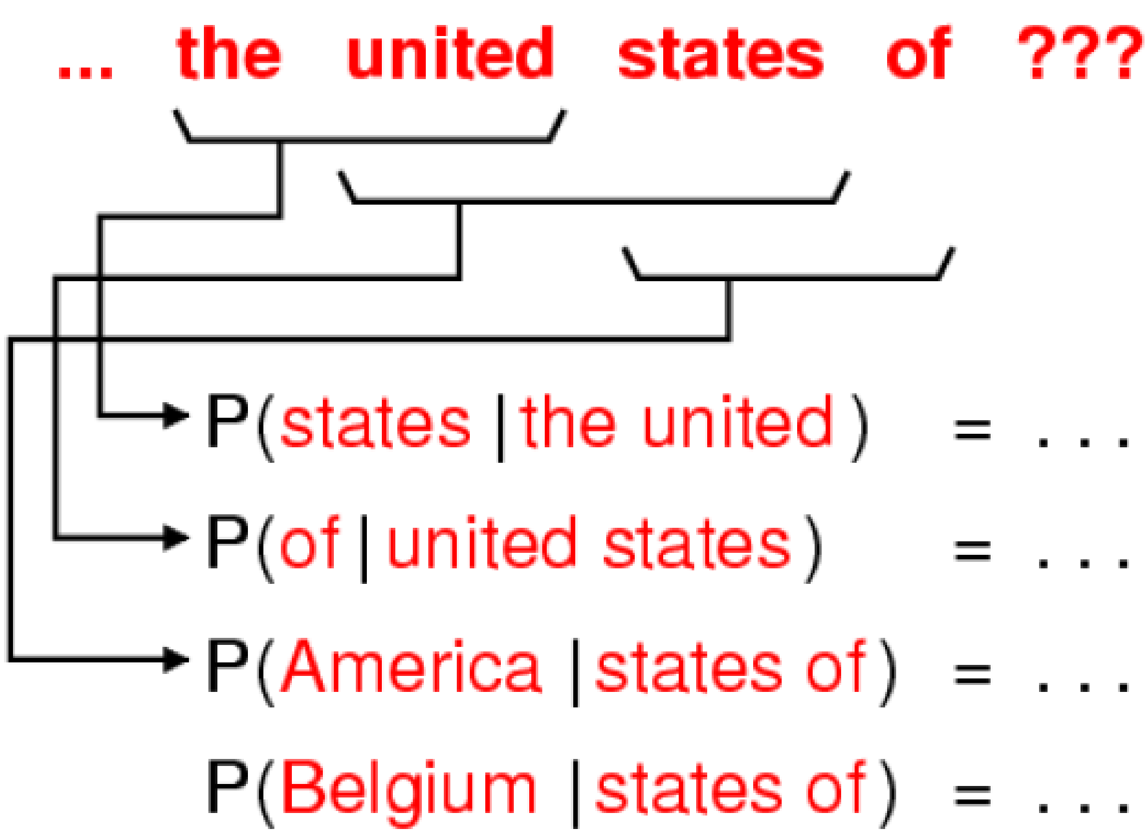
Picture by S.Renals

N-gram language model

- Stochastic model of the **relations between words**
  - Which words often occur close to each other?
- The model **predicts the probability distribution of the next word** given the previous ones
- A conditional probability of word given its **context**
- Estimated from a large text corpus (count the contexts!)
- Smoothing and pruning required to learn compact long-span models from **sparse training data**

N-gram models

- E.g. trigram = 3-gram:
- Word occurrence depends only on its immediate short context
- A conditional probability of word given its context
- Estimated from a large text corpus (count the contexts!)



Estimation of N-gram model

$$P(w_i \mid w_j) = \frac{c(w_j, w_i)}{c(w_j)}$$

$$\frac{c(\text{“eggplant stew”})}{c(\text{“eggplant”})}$$

- Bigram example:
  - Start from a **maximum likelihood estimate**
  - probability of *P(“stew” | “eggplant”)* is computed from **counts** of *“eggplant stew”* and *“eggplant”*

Picture by B.Pellom

Data from Berkeley restaurant corpus (Jurafsky & Martin, 2000 "Speech and language processing").

	I	want	to	eat	Chinese	food	lunch
I	8	1087	0	13	0	0	0
want	3	0	786	0	6	8	6
to	3	0	10	860	3	0	12
eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
food	19	0	17	0	0	0	0
lunch	4	0	0	0	0	1	0

I	3437
want	1215
to	3256
eat	938
Chinese	213
food	1506
lunch	459

Uni-gram counts

Calculate missing bi-gram probabilities

	I	want	to	eat	Chinese	food	lunch
I	.0023	X	0	.0038	0	0	0
want	.0025	0	.65	0	.0049	.0066	X
to	.00092	0	.0031	.26	X	0	.0037
eat	0	0	.0021	0	.020	.0021	.055
Chinese	.0094	0	0	0	0	.056	.0047
food	.013	0	.011	0	0	0	0
lunch	.0087	0	0	0	0	.0022	0

Data from Berkeley restaurant corpus (Jurafsky & Martin, 2000 "Speech and language processing").

	I	want	to	eat	Chinese	food	lunch
I	8	1087	0	13	0	0	0
want	3	0	786	0	6	8	6
to	3	0	10	860	3	0	12
eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
food	19	0	17	0	0	0	0
lunch	4	0	0	0	0	1	0

Uni-gram counts

I	3437
want	1215
to	3256
eat	938
Chinese	213
food	1506
lunch	459

1087 / 3437=.32

Calculate missing bi-gram probabilities

	I	want	to	eat	Chinese	food	lunch
I	.0023	.32	0	.0038	0	0	0
want	.0025	0	.65	0	.0049	.0066	X
to	.00092	0	.0031	.26	X	0	.0037
eat	0	0	.0021	0	.020	.0021	.055
Chinese	.0094	0	0	0	0	.056	.0047
food	.013	0	.011	0	0	0	0
lunch	.0087	0	0	0	0	.0022	0

Data from Berkeley restaurant corpus (Jurafsky & Martin, 2000 "Speech and language processing").

	I	want	to	eat	Chinese	food	lunch
I	8	1087	0	13	0	0	0
want	3	0	786	0	6	8	6
to	3	0	10	860	3	0	12
eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
food	19	0	17	0	0	0	0
lunch	4	0	0	0	0	1	0

Uni-gram counts

I	3437
want	1215
to	3256
eat	938
Chinese	213
food	1506
lunch	459

1087 / 3437=.32

3 / 3256 = .00092

Calculate missing bi-gram probabilities

	I	want	to	eat	Chinese	food	lunch
I	.0023	.32	0	.0038	0	0	0
want	.0025	0	.65	0	.0049	.0066	X
to	.00092	0	.0031	.26	.00092	0	.0037
eat	0	0	.0021	0	.020	.0021	.055
Chinese	.0094	0	0	0	0	.056	.0047
food	.013	0	.011	0	0	0	0
lunch	.0087	0	0	0	0	.0022	0

Data from Berkeley restaurant corpus (Jurafsky & Martin, 2000 "Speech and language processing").

	I	want	to	eat	Chinese	food	lunch
I	8	1087	0	13	0	0	0
want	3	0	786	0	6	8	6
to	3	0	10	860	3	0	12
eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
food	19	0	17	0	0	0	0
lunch	4	0	0	0	0	1	0

Uni-gram counts

I	3437
want	1215
to	3256
eat	938
Chinese	213
food	1506
lunch	459

1087 / 3437=.32

3 / 3256 = .00092

6 / 1215 = .0049

Calculate missing bi-gram probabilities

	I	want	to	eat	Chinese	food	lunch
I	.0023	.32	0	.0038	0	0	0
want	.0025	0	.65	0	.0049	.0066	.0049
to	.00092	0	.0031	.26	.00092	0	.0037
eat	0	0	.0021	0	.020	.0021	.055
Chinese	.0094	0	0	0	0	.056	.0047
food	.013	0	.011	0	0	0	0
lunch	.0087	0	0	0	0	.0022	0

Estimation of N-gram model

$$P(w_i \mid w_j) = \frac{c(w_j, w_i)}{c(w_j)}$$

$$\frac{c(\text{“eggplant stew”})}{c(\text{“eggplant”})}$$

- Bigram example:
  - Start from a **maximum likelihood estimate**
  - probability of *P(“stew” | “eggplant”)* is computed from **counts** of *“eggplant stew”* and *“eggplant”*
  - works well for frequent bigrams P(“want”|”I”) = 1087 / 3437 = 0.32
  - why not for rare bigrams? P(“Chinese”|”to”) = 3 / 3256 = 0.00092

Picture by B.Pellom

Exercise 2A: Where to use language models?

- Go in breakout rooms and discuss this topic
- Submit notes from your discussion in **MyCourses > Lectures > Lecture 2A exercise return box:**
  - List as many potential applications for statistical language models as you can!
  - Typically these are tasks where you need the probability or to find the most probable word or sentence given some background information

Some applications of SLMs

1. Spelling correction, text input
2. Optical character recognition, e.g. scanning old books
3. Automatic speech recognition
4. Statistical machine translation
5. Text-to-speech
6. Automatic question answering
7. Chatbots

Data sparsity

- Words and many other linguistic units follow a **power-law distribution:**
  - Zipf's law: kth frequent word occurs ∝ 1/k
  - "Long tail": few frequent words, lots of very rare words
- E.g. within the first 1.5 million words 23% subsequent trigrams were previously unseen (IBM laser patent text corpus)
- Maximum likelihood estimate overestimates frequencies of n-gram that occurred rarely, and underestimates those that did not occur at all. (why?)
- One needs a systematic approach to assign some non-zero probability to unseen words and sequences. This is called **smoothing**.

Zero probability problem

- If an N-gram is not seen in the corpus, it will get probability = 0
- The higher N, the sparser data, and the more zero counts there will be
- 20K words => 400M 2-grams => 8000G 3-grams, so even the largest corpora have MANY zero counts!

Solutions:

- **Equivalence classes**: Cluster several similar n-grams together to reach higher counts
- **Smoothing**: Redistribute some probability mass from seen N-grams to unseen ones

Equivalence classes

- Divide features (e.g. words) into equivalence classes a.k.a. **bins**
- Assume equal statistical properties within a bin
- Estimate a SLM for the bin as a whole
- The more bins, the more data is needed for model estimation
- The fewer bins, the lower prediction accuracy, because the model becomes too general

Ways to form classes

- Transforming **inflected word forms** into the baseform: ’saunan’, ’saunalle’, ’saunojemme’, etc. → ’sauna’
- Grouping by **part-of-speech** tags (the same syntactic role: *noun, verb*, etc)
- Grouping by **semantics** (a similar meaning)

Important is that the words in a bin should really behave similarly! E.g. *february, may, august*

Ways to use classes

- using equivalence classes only for previous words (history):

  $p(w_i \mid w_{i-2}, w_{i-1}) = p(w_i \mid \mathbf{t(w_{i-2}, w_{i-1})})$

- using class-based n-gram models:

  $p(w_i \mid w_{i-2}, w_{i-1}) = p(\mathbf{t(w_i)} \mid \mathbf{t(w_{i-2}, w_{i-1})})$
  $\times\, p(w_i \mid \mathbf{t(w_i)}, \ldots)$

- determine bin-specific interpolation weights for model combination (Broman and Kurimo, 2005)

Combining estimators

- So far, the probability was estimated for all n-grams of a particular length
- How about improving the estimate using shorter sequences that are more frequent?
- The motivation is further smoothing of the estimates by **combining different information sources**.
- The **additional models** could also be other n-grams trained on **different data**, e.g. background models vs topical models

Backing-off

- **In principle**: Look for the most specific model that gives sufficient information from the current context
- **In practice**: Back off from using (too) long contexts to shorter ones that have more samples in the corpus.

Smoothing methods

1. **Add-one**: Add 1 to each count and normalize => gives too much probability to unseen N-grams
2. **(Absolute) discounting**: Subtract a constant from all counts and redistribute this to unseen ones using N-1 gram probs and back-off (normalization) weights
3. **Witten-Bell smoothing**: Use the count of things seen once to help to estimate the count of unseen things
4. **Good Turing smoothing**: Estimate the rare n-grams based on counts of more frequent counts
5. Best: **Kneser-Ney smoothing**: Instead of the number of occurrences, weigh the back-offs by the **number of contexts** the word appears in
6. Instead of only back-off cases, **interpolate** all N-gram counts with N-1 counts

Add-1 smoothing

$$c_i^* = (c_i + 1)\frac{N}{N+V}$$

Probability p = c / N :

$$p_i^* = \frac{c_i + 1}{N+V}$$

Ci*,Ci: new count, old c

N : Num of tokens

T : Num of types (seen)

Z : Num of types (unseen)

V : Total vocab size

	I	want	to	eat	Chinese	food	lunch
I	9	1088	1	14	1	1	1
want	4	1	787	1	7	9	7
to	4	1	11	861	4	1	13
eat	1	1	3	1	20	3	53
Chinese	3	1	1	1	1	121	2
food	20	1	18	1	1	1	1
lunch	5	1	1	1	1	2	1

Figure 6.6 Add-one Smoothed Bigram counts for 7 of the words (out of 1616 total word types) in the Berkeley Restaurant Project corpus of ~10,000 sentences.

$$c_i^* = (c_i + 1)\frac{N}{N+V}$$

Probability p = c / N :

$$p_i^* = \frac{c_i + 1}{N+V}$$

N : Num of tokens

T : Num of types (seen)

Z : Num of types (unseen)

V : Total vocab size

$$c_i^* = \begin{cases} \frac{T}{Z}\frac{N}{N+T}, & \text{if } c_i = 0 \\ c_i \frac{N}{N+T}, & \text{if } c_i > 0 \end{cases}$$

	I	want	to	eat	Chinese	food	lunch
I	9	1088	1	14	1	1	1
want	4	1	787	1	7	9	7
to	4	1	11	861	4	1	13
eat	1	1	3	1	20	3	53
Chinese	3	1	1	1	1	121	2
food	20	1	18	1	1	1	1
lunch	5	1	1	1	1	2	1

Figure 6.6 Add-one Smoothed Bigram counts for 7 of the words (out of 1616 total word types) in the Berkeley Restaurant Project corpus of ~10,000 sentences.

	I	want	to	eat	Chinese	food	lunch
I	8	1060	.062	13	.062	.062	.062
want	3	.046	740	.046	6	8	6
to	3	.085	10	827	3	.085	12
eat	.075	.075	2	.075	17	2	46
Chinese	2	.012	.012	.012	.012	109	1
food	18	.059	16	.059	.059	.059	.059
lunch	4	.026	.026	.026	.026	1	.026

Figure 6.9 Witten-Bell smoothed bigram counts for 7 of the words (out of 1616 total word types) in the Berkeley Restaurant Project corpus of ~10,000 sentences.

Good-Turing smoothing

- How to compute the **probability of an unseen event,** e.g. an out-of-vocabulary word?
- Idea invented by Alan Turing during World War 2 when he was working to break German cipher
- Published later by his student (Good, 1953)
- Set:
  - N = Num of words
  - N_c = Num of words that occur c-times (freq. of freq.)
- Estimate prob of unseen things = N_1/N
- Estimate count of things seen once = (c+1)*N_2/N_1
- Smoothed count c* for all c:

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

Exercise 2B: Good-Turing smoothing

- **Watch a video** where Prof. Jurafsky (Stanford) explains Good-Turing smoothing (between 02:00 – 08:45)
  - *Click: http://www.youtube.com/watch?v=GwP8gKa-ij8*
  - *Or search:"Good Turing video Jurafsky"*
- Go in breakout rooms and submit answers for 3 questions in **MyCourses > Lectures > Lecture 2B exercise return box**:

1. Estimate the prob. of catching next any new fish species, if you already got: 5 perch, 2 pike, 1 trout, 1 zander and 1 salmon?
2. Estimate the prob. of catching next a salmon?
3. What may cause practical problems when applying Good-Turing smoothing for rare words in large text corpora?

Hints for solving the exercise

1. Estimate the prob of unseen things using the prob of things seen only once N_1/N
2. The counts must be smoothed. The new count for things seen once is (c+1)*N_2/N_1
3. What if N_c = 0 for some c?

Estimation of N-gram model

$$P(w_i \mid w_j) = \frac{c(w_j, w_i)}{c(w_j)}$$

$$\frac{c(\text{"eggplant stew"})}{c(\text{"eggplant"})}$$

- Bigram example:
  - Start from a **maximum likelihood estimate**
  - probability of *P("stew" | "eggplant")* is computed from **counts** of *"eggplant stew"* and *"eggplant"*
  - works well for frequent bigrams

Picture by B.Pellom

Backing off

$$P(w_i \mid w_j) = \frac{c(w_j, w_i)}{c(w_j)} \qquad \text{if } c(w_j, w_i) > c$$
$$= P(w_i) b_{w_j} \qquad \textit{otherwise}$$

- Divide the room of rare bigrams, e.g. *"eggplant francisco"*, in proportion to the unigram ***P("francisco")***
- The sum of all these rare bigrams *"eggplant [word j]"* is ***b("eggplant")*** which is called the **back-off weight**

Picture by B.Pellom

Absolute discounting and backing off

$$P(w_i \mid w_j) = \frac{c(w_j, w_i) - D}{c(w_j)} \quad \text{if } c(w_j, w_i) > c$$
$$= P(w_i) b_{w_j} \qquad \textit{otherwise}$$

- If bigram is common: Subtract constant *D* from the count
- If not: Back off to the unigram probability normalized by the back-off weight
- Similarly back off all rare N-grams to N-1 grams

Picture by B.Pellom

Kneser-Ney smoothing

$$P(w_i \mid w_j) = \frac{c(w_j, w_i) - D}{c(w_j)} \quad \text{if } c(w_j, w_i) > c$$

$$= \mathbf{V}(w_i) b_{w_j} \qquad \textit{otherwise}$$

- Instead of the number of occurrences, weigh the back-offs by the **number of contexts** ***V****(word)* the word appears in:
  - In this case the context is the previous word, thus, how many different previous words the corpus has for that word
  - E.g. *P(Stew | EggPlant)* is high, because stew occurs in many contexts
  - But *P(Francisco | EggPlant)* is low, because Francisco is common, but only in "San Francisco"

Picture by B.Pellom

Smoothing by interpolation

$$P(w_i \mid w_j) = \frac{c(w_j, w_i) - D}{c(w_j)} + P(w_i) b_{w_j}$$

- Like backing off, but always compute the probability as a **linear combination** (weighted average) with lower order (N-1)gram probabilities
- Improves the probabilities of rare N-grams
- Discounts (D) (and interpolation weights) can be separately optimized for each N using a held-out data

N-gram example

eggplant X)	1G freq	1G prob	2G freq	2G prob
X = stew	10	0.1	0	0
sue	20	0.2	0	0
san	40	0.4	0	0
francisco	30	0.3	0	0
SUM	100	1	0	0

10/100

$$P(w_i \mid w_j) = \frac{c(w_j, w_i)}{c(w_j)}$$

Absolute discounting

eggplant X)	1G freq	1G prob	2G freq	2G prob	D=0.50 discount
X = stew	10	0.1	0	0	
sue	20	0.2	0	0	
san	40	0.4	0	0	
francisco	30	0.3	0	0	
SUM	100	1	0	0	0.5

$$P(w_i \mid w_j) = \frac{c(w_j, w_i) - D}{c(w_j)} \quad \text{if } c(w_j, w_i) > c$$

(c=0, D=0.5 selected)

Back-off

D=0.50

eggplant X)	1G freq	1G prob	2G freq	2G prob	discount	Abs back-off	normalize
X = stew	10	0.1	0	0		0.1	0.05
sue	20	0.2	0	0		0.2	0.1
san	40	0.4	0	0		0.4	0.2
francisco	30	0.3	0	0		0.3	0.15
SUM	100	1	0	0	0.5	1	0.5

$$
\begin{aligned}
P(w_i \mid w_j) &= \frac{c(w_j, w_i) - D}{c(w_j)} && \text{if } c(w_j, w_i) > c \\
&= P(w_i) b_{w_j} && otherwise
\end{aligned}
$$

Back-off

							D=0.50	
eggplant X)	1G freq	1G prob	2G freq	2G prob	discount	Abs back-off	normalize
X = stew	10	0.1	0	0		0.1	0.05
sue	20	0.2	0	0		0.2	0.1
san	40	0.4	0	0		0.4	0.2
francisco	30	0.3	0	0		0.3	0.15
SUM	100	1	0	0	0.5	1	0.5

$$P(w_i \mid w_j) = \frac{c(w_j, w_i) - D}{c(w_j)} \quad \text{if } c(w_j, w_i) > c$$

$$= P(w_i) b_{w_j} \qquad otherwise$$

0.1/1.0*0.5

Absolute discounting and back-off

(eggplant X)	1G freq	2G freq	Abs back-off	normalize
X = stew	10	0	0.1	0
sue	20	0	0.2	0
san	40	0	0.4	0
francisco	30	0	0.3	0
SUM	100	0	1	0

$$P(w_i \mid w_j) = \frac{c(w_j, w_i) - D}{c(w_j)} \quad \text{if } c(w_j, w_i) > c$$

$$= P(w_i) b_{w_j} \quad otherwise$$

(c=0, D=0.5 selected)

Kneser-Ney smoothing

(eggplant X)	1G freq	2G freq	Abs back-off	normalize	#contexts
X = stew	10	0	0.1	0	10
sue	20	0	0.2	0	5
san	40	0	0.4	0	3
francisco	30	0	0.3	0	1
SUM	100	0	1	0	19

$$P(w_i \mid w_j) = \frac{c(w_j, w_i) - D}{c(w_j)} \quad \text{if } c(w_j, w_i) > c$$

$$= \mathbf{V}(w_i) b_{w_j} \qquad otherwise$$

(c=0, D=0.5 selected)

Kneser-Ney smoothing

(eggplant X)	1G freq	2G freq	Abs back-off	normalize	#contexts	KN back-off
X = stew	10	0	0.1	0.05	10	0.26
sue	20	0	0.2	0.1	5	0.13
san	40	0	0.4	0.2	3	0.08
francisco	30	0	0.3	0.15	1	0.03
SUM	100	0	1	0.5	19	0.5

$$P(w_i \mid w_j) = \frac{c(w_j, w_i) - D}{c(w_j)} \quad \text{if } c(w_j, w_i) > c$$

$$= \mathbf{V}(w_i) b_{w_j} \quad otherwise$$

(c=0, D=0.5 selected)

10/19*0.5

Weaknesses of N-grams

- Skips long-span dependencies:
  - “The **girl** that I met in the train **was** ...”
- Too dependent on word order:
  - “dog chased **cat**”: “koira jahtasi **kissaa**” ~ “**kissaa** koira jahtasi”
- Dependencies directly between words, instead of latent variables, e.g. word categories

Some model variants

- Variable-length n-gram, aka. **Varigram**:
  - Span depends on particular context, optimized for the data, e.g. [Siivola, 2007]
  - Especially useful for short units (letters, morphemes)
- **Class-based** n-gram, e.g. [Brown, 1992]:
  - Cluster words into classes, find class sequences
  - Reduces sparsity, model size, and accuracy
- **Bayesian** n-gram:
  - Computationally demanding
  - Kneser-Ney smoothing approximates hierarchical Pitman-Yor process model [Goldwater, 2006; Teh, 2006]

Sources and further reading

- Manning, C. D. and Schütze, H. (1999). Foundations of Statistical Natural Language Processing. The MIT Press. (Chapter 6)
- Jurafsky, D. and Martin, J. H. (2008). Speech and Language Processing. Prentice Hall. 2nd edition. (Chapter 4)
- Chen, S. F. and Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. Computer Speech and Language, 13(4):359–393.
- Goodman, J. T. (2001). A bit of progress in language modeling - extended version. Technical Report MSR-TR-2001-72, Microsoft Research.
- Virpioja, S. (2012). Learning Constructions of Natural Language: Statistical Models and Evaluations. Aalto University, Doctoral dissertations 158/2012. (Sections 4.1–4.3)
- Varjokallio, M. (2020). Improving very large vocabulary language modeling and decoding for speech recognition in morphologically rich languages. Aalto University, Doctoral dissertations 208/2020.(Section 4.1)

Other language modeling (LM) approaches

- **Maximum-entropy** LM [1]
  - Combines different knowledge sources into a single model
  - Good for adaptation [5]
- **Continuous-space** LM (a.k.a. Neural Network LMs)
  - Map words to continuous-valued vectors and models them using e.g. neural networks [2,3]
  - State-space models can use indefinitely long contexts, such as in Recurrent Neural Networks [4]
- **Cache** models and **Topic** models

(1)R.Rosenfeld. A maximum entropy approach to adaptive statistical language modelling. Computer Speech and Language, 2007.

(2)Y.Bengio & al. A neural probabilistic language model. Journal of Machine Learning Research, 2003.

(3)V.Siivola, A.Honkela. A State-Space Method for Language Modeling, Proc. ASRU 2003.

(4)T.Mikolov & al. Recurrent neural network based language model. Proc. Interspeech 2010.

(5)T.Alumäe, M.Kurimo. Domain adaptation of maximum entropy language models. Proc. ACL 2010.

Maximum entropy LMs

$$P(x|h) = \frac{e^{\sum_i \lambda_i f_i(x,h)}}{\sum_{x'} e^{\sum_j \lambda_j f_j(x',h)}}$$

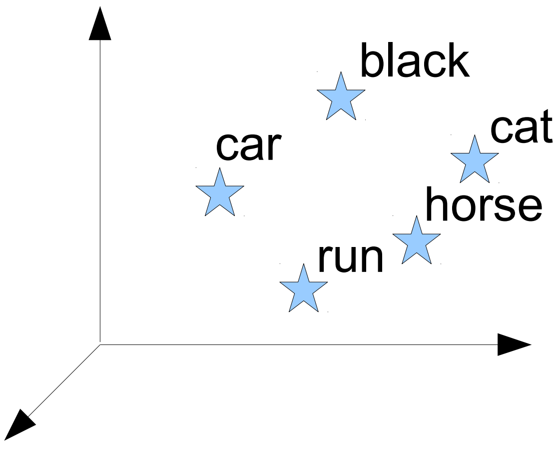
- Represents dependency information by a weighted sum of features f(x,h)
- Features can be e.g. n-gram counts
- Alleviates the data sparsity problem by smoothing the feature weights (lambda) towards zero
- The weights can be adapted in more flexible ways than n-grams
  - Adapting only those weights that significantly differ from a large background model (1)
- Normalization is computationally hard, but can be approximated effectively

---

(1) T.Alumäe, M.Kurimo. Domain adaptation of maximum entropy language models. Proc. ACL 2010.

Mapping words into continuous space

- Map words into a continuous vector space to learn a distributed representation known as *word embedding*
- The goal is to use a vector space that keeps similarly behaving words near each other



- Words can be clustered by context, e.g. n-gram probabilities
  - *word2vec* [1] is one widely used option
  - Other embeddings to reflect various contextual properties
- Set of words can be represented by a sum of the vectors
- N-gram can be represented by a sequence of vectors

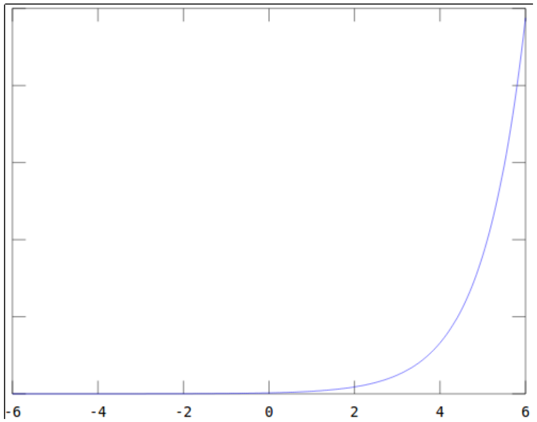
(1) T.Mikolov et al. Efficient Estimation of Word Representations in Vector Space. 2013. ArXiv:1301.3781.

Continuous space LMs

- Alleviates the data sparsity problem by representing words in a distributed way
- Various algorithms can be used to learn the most efficient and discriminative representations and classifiers
- The most popular family of algorithm is called (Artificial) **Neural Networks (NN)**
  - can learn very complex functions by combining simple computation units in a hierarchy of non-linear layers
  - Fast in action, but training takes a lot of time and labeled training data
- Can be seen as a non-linear multilayer generalization of the maximum entropy model

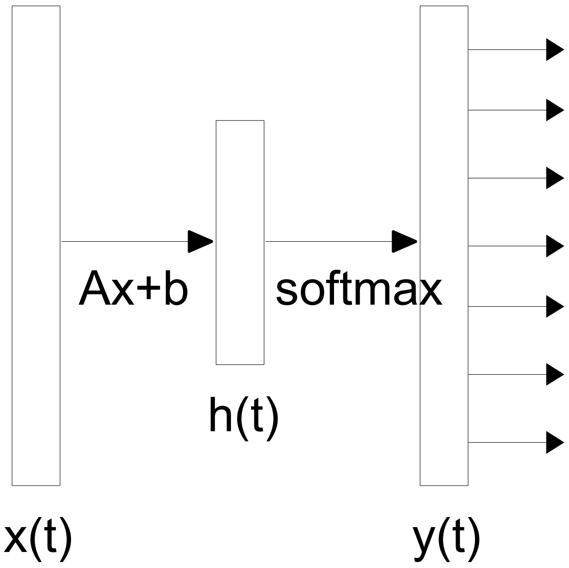
A <u>simple</u> bigram NN LM

- Outputs the *probability of next word* y(t) given the previous word x(t)
- **Input layer** maps the previous word as a vector x(t)
- **Hidden layer** has a linear transform h(t) = Ax(t) + b to compute a representation of *linear distributional features*
- **Output layer** maps the values by y(t) = softmax (h(t)) *to range (0,1) that add up to 1*
- Resembles a bigram Maximum entropy LM



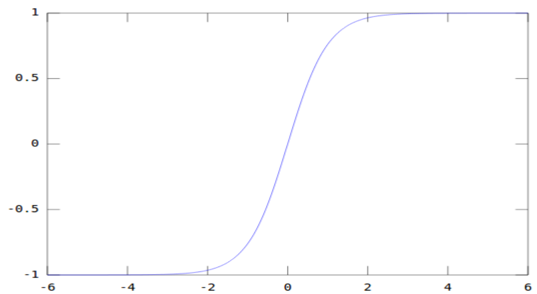
Softmax:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

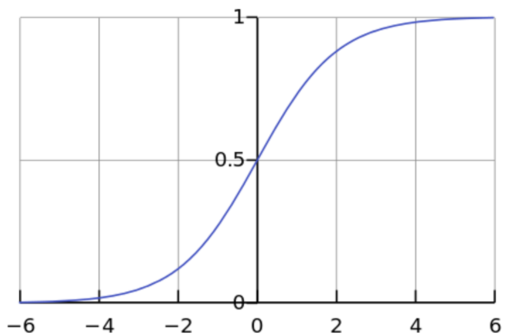


A non-linear bigram NN LM

- The only difference to the simple NN LM is that the hidden layer h(t) now includes a non-linear function $h(t) = U(Ax(t) + b)$
- Can learn more complex feature representations
- Common examples of non-linear functions U:

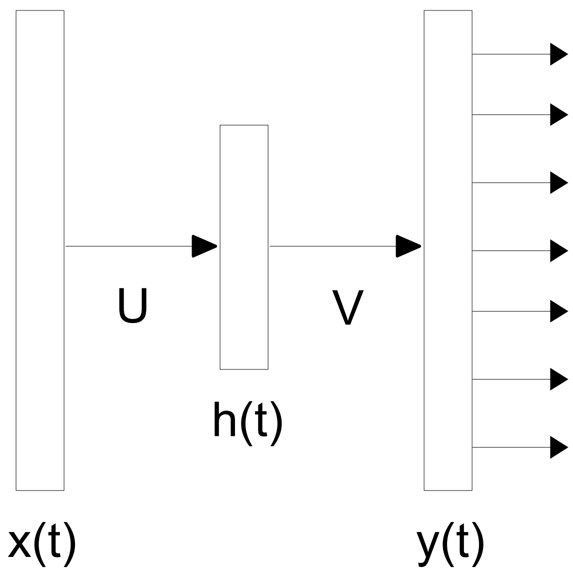


$$U(t) = \tanh(t)$$



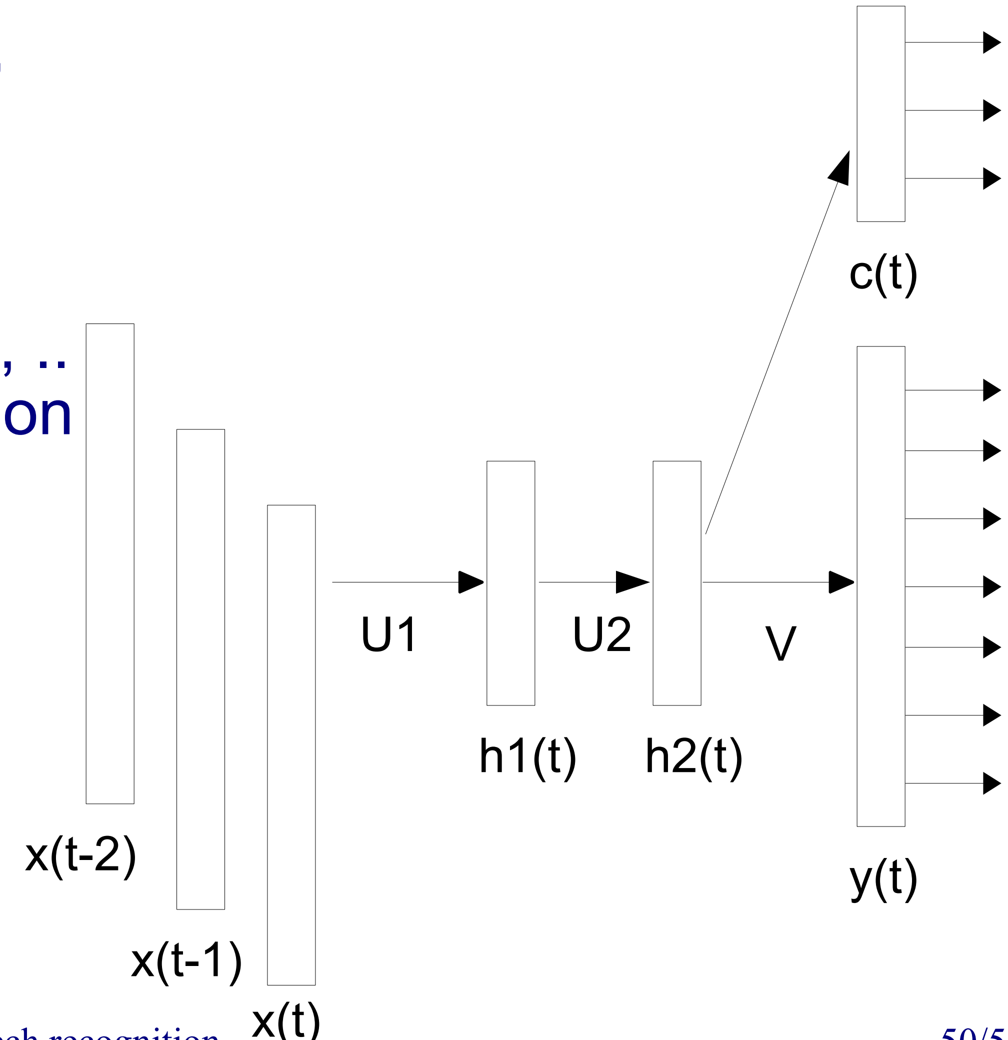
Sigmoid

$$U(t) = \frac{1}{1 + e^{-t}}$$



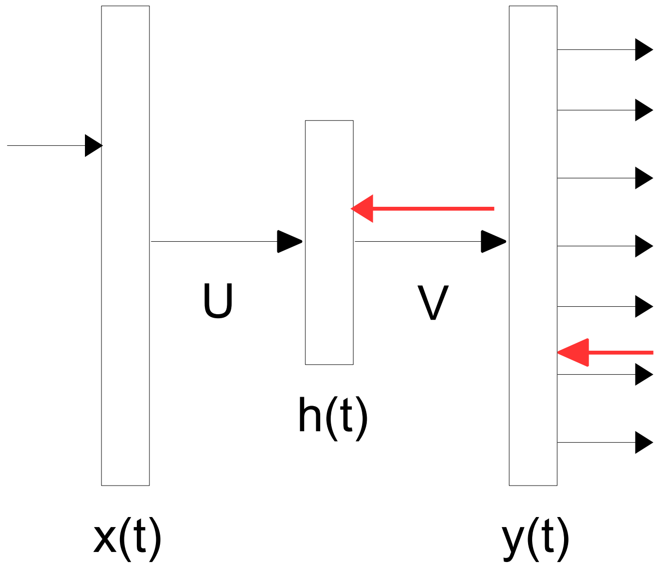
Common NN LM extensions

- ***Input layer*** is expanded *over several previous words* x(t-1), x(t-2), .. to learn richer representations
- *Deep neural networks* have *several **hidden layers*** h1, h2, .. to learn to represent information at several hierarchical levels
- Can be scaled to a very large vocabulary by training also *a class-based **output layer** c(t)*



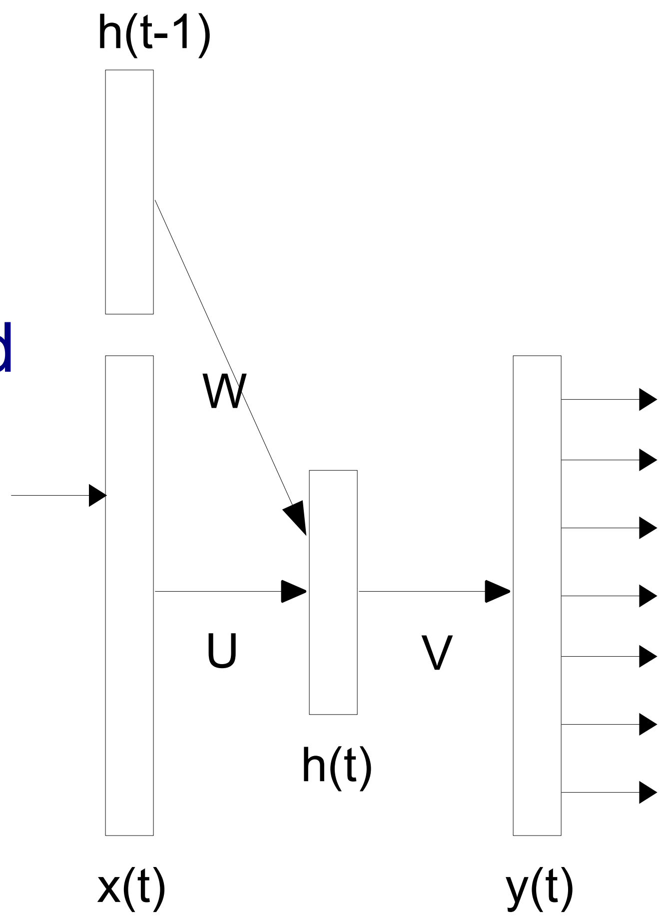
NN LM training

- Supervised training minimizes the **output errors** by training the weights for V by *stochastic gradient descend*
- Propagate the output **error to hidden layer** to train the weights for U
- In practice, a deep NN will require more complex training procedures, since the gradients *vanish* quickly



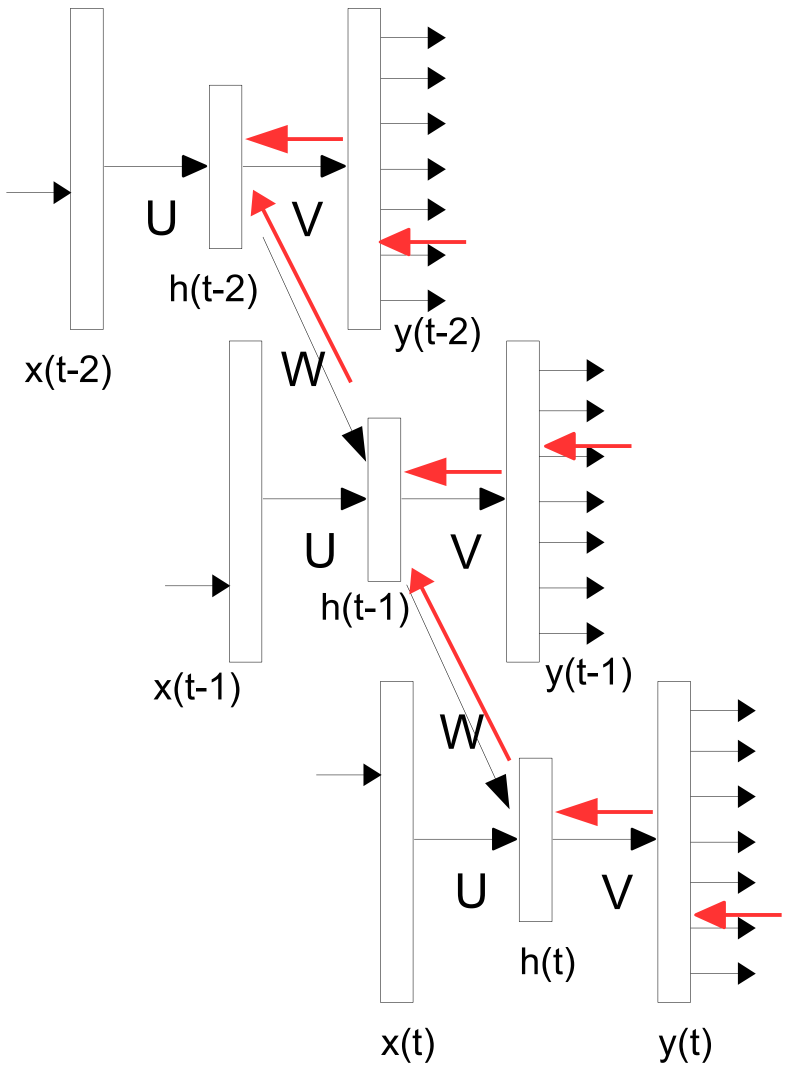
Recurrent Neural Network (RNN) LM

- **Looks like** a bigram NNLM
- **But**, takes an additional input from the hidden layer of the *previous time* step
- Hidden layer becomes a compressed representation of the word history
- Can learn to represent unlimited memory, in theory
- Currently, the state-of-the-art in LMs



RNN LM training

- Minimizes the output error by training the weights by *stochastic gradient descend*
- Propagates the output error to all *layers and time* steps (called *backpropagation through time)* to train the hidden layer
- **Looks now like** a very deep neural network with shared weights U and W



References (all)

- Markov, A. A. (1913). An example of statistical investigation of the text Eugene Onegin concerning the connection of samples in chains. (In Russian.) Bulletin of the Imperial Academy of Sciences of St. Petersburg 7(3):153–162.
- Shannon, C. E. (1948). A mathematical theory of communication. Bell System Technical Journal, 27:379–423, 623–656.
- Good, I.J. (1953). The population frequencies of species and the estimation of population parameters. Biometrika 40 (3–4): 237–264
- Kohonen, T. (1986). Dynamically Expanding Context, with application to the correction of symbol strings in the recognition of continuous speech", Proc. ICPR 1986, pp.1148-1151
- Ritter, H. and Kohonen, T. (1989). Self-organized semantic maps. Biol. Cybern. 61: 241-254
- Kohonen, Kaski, Lagus, Honkela (1996). Very large two-level SOM for the browsing of newsgroups. Proc. ICANN96.
- Kneser, R. and Kney, H. (1995). Improved backing-off for m-gram language modeling. IEEE Trans. ASSP, 1:181–184.

References (cont'd)

- Brown, P. F., DellaPietra, V. J., deSouza, P. V., Lai, J. C., and Mercer, R. L. (1992). Class-based n-gram models of natural language. Computational Linguistics, 18(4):467–479.
- Siivola, V., Hirsimäki, T. and Virpioja, S. (2007). On Growing and Pruning Kneser-Ney Smoothed N-Gram Models. IEEE Trans. ASLP, 15(5):1617-1624.
- Siivola, V., Pellom, B. (2005). Growing an n-gram model, Proc. Interspeech'05, pp. 1309-1312.
- Goldwater, S., Griffiths, T., and Johnson, M. (2006). Interpolating between types and tokens by estimating power-law generators. In Advances in NIPS 18, pp. 459–466. MIT Press.
- Teh, Y. W. (2006). A hierarchical Bayesian language model based on Pitman-Yor processes. Proc. ACL 2006, pp. 985–992.
- Roark, B. (2001). Probabilistic top-down parsing and language modeling. Computational Linguistics, 27(2):249–276.
- Creutz ,M., Lagus, K. (2003). Unsupervised discovery of morphemes. Proc. Workshop on Morphological and Phonological Learning of ACL-02,pp.21–30

References (cont'd)

- Rosenfeld, R. (1996). A maximum entropy approach to adaptive statistical language modelling. Computer Speech and Language, 10(3):187–228.
- Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. Journal of Machine Learning Research, 3:1137–1155.
- Siivola, V., Honkela, A. (2003). A State-Space Method for Language Modeling", IEEE Workshop on Automatic Speech Recognition and Understanding, pp 548-553.
- Mikolov, T., Karafiat, M., Burget, L., Cernocky, J., and Khudanpur, S. (2010). Recurrent neural network based language model. Proc. Interspeech 2010, pp. 1045–1048
- Alumäe, T., Kurimo, M. (2010) Domain adaptation of maximum entropy language models. Proc. ACL 2010.
- Broman, S., Kurimo, M. (2005). Methods for combining language models in speech recognition. Proc. Interspeech 2005, pp. 1317–1320.
- Hirsimäki, Creutz, Siivola, Kurimo, Virpioja and Pylkkönen (2006). Unlimited vocabulary speech recognition with morph language models applied to Finnish. Computer Speech and Language 20(4):515--541

Feedback

Go to **MyCourses > Lectures > Feedback for Lecture 2** and fill in the form.

Some of the feedback from the previous week:

+ The lecture was clear and at an appropriate pace

+ The small group thing was okey. There was some talking. Not sure if 10 min is required for that

+ Aalto research stuff highlighting concept. First time saw this in any lecture..

- I didn't fully grasp the project work goals and practicalities

- Maybe some video presentation of available techniques would make the lecture even more thrilling

- I think you could end the break out rooms and then announce the break

- Why is only half of the recorded lecture available?

Thanks for all the valuable feedback!