Unsupervised Language Learning: Representation Learning for NLP

Katia Shutova

ILLC
University of Amsterdam

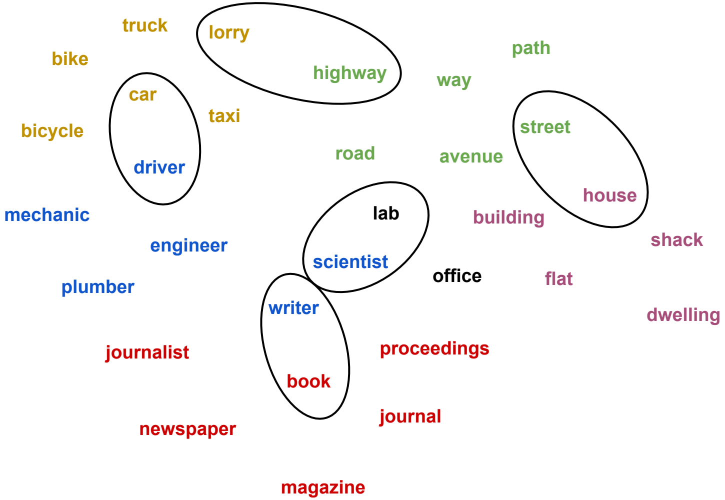
4 April 2018

Lecture 2: Semantics with dense vectors & compositional semantics

Outline.

Clustering nouns

Clustering nouns



Outline.

Word clustering (finishing off)

Semantics with dense vectors

Compositional semantics

Compositional distributional semantics

Semantic composition in recurrent neural networks

Distributional semantic models

1. Count-based models:
   - Explicit vectors: dimensions are elements in the context
   - **long sparse** vectors with **interpretable** dimensions
2. Prediction-based models:
   - Train a model to predict plausible contexts for a word
   - learn word representations in the process
   - **short dense** vectors with **latent** dimensions

Sparse vs. dense vectors

Why dense vectors?

- easier to use as features in machine learning (less weights to tune)
- may generalize better than storing explicit counts
- may do better at capturing synonymy:
  - e.g. *car* and *automobile* are distinct dimensions in count-based models
  - will not capture similarity between a word with *car* as a neighbour and a word with *automobile* as a neighbour

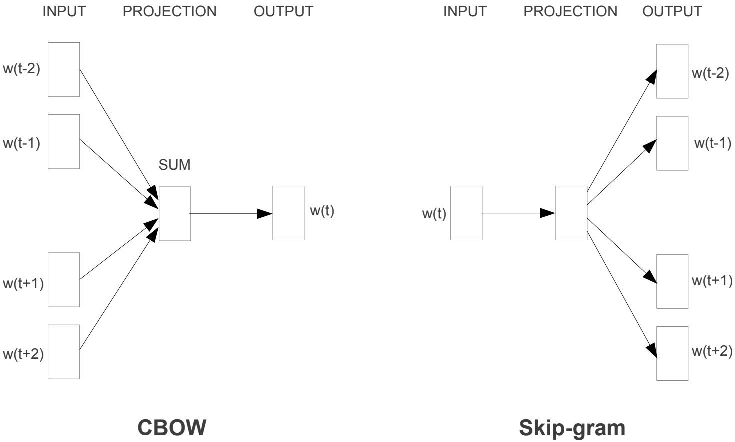
Prediction-based models

Mikolov et. al. 2013. *Efficient Estimation of Word Representations in Vector Space*.

word2vec: **Skip-gram** and **CBOW** (continuous bag of words)

- inspired by work on neural language models
- train a neural network to predict neighboring words
- learn dense embeddings for the words in the training corpus in the process

Skip-gram vs. CBOW



Slide credit: Tomas Mikolov

Skip-gram

Intuition: words with similar meanings often occur near each other in texts

Given a word w_t:

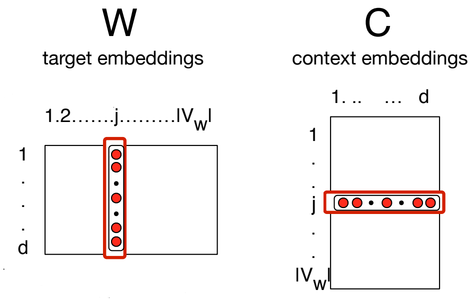
- Predict each neighbouring word
  - in a context window of $2L$ words
  - from the current word.
- For $L = 2$, we predict its 4 neighbouring words:

$$[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$$

Skip-gram: Parameter matrices

Learn 2 embeddings for each word $w_j \in V_w$:

- **word embedding** v, in word matrix W
- **context embedding** c, in context matrix C



Skip-gram: Setup

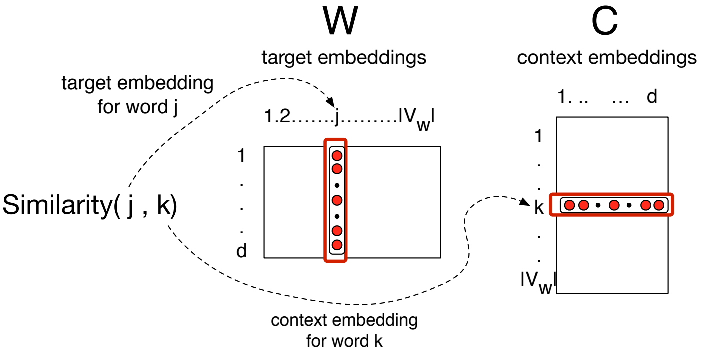
- Walk through the corpus pointing at word $w(t)$, whose index in the vocabulary is j — we will call it w_j
- our goal is to predict $w(t+1)$, whose index in the vocabulary is k — we will call it w_k
- to do this, we need to compute

$$p(w_k|w_j)$$

- Intuition behind skip-gram: to compute this probability we need to compute similarity between w_j and w_k

Skip-gram: Computing similarity

Similarity as dot-product between the target vector and context vector



Slide credit: Dan Jurafsky

Skip-gram: Similarity as dot product

- Remember cosine similarity?

$$cos(v1, v2) = \frac{\sum v1_k * v2_k}{\sqrt{\sum v1_k^2} * \sqrt{\sum v2_k^2}} = \frac{v1 \cdot v2}{||v1||||v2||}$$

It's just a normalised dot product.

- Skip-gram: Similar vectors have a high dot product

$$Similarity(c_k, v_j) \propto c_k \cdot v_j$$

Skip-gram: Compute probabilities

- Compute similarity as a dot product

$$Similarity(c_k, v_j) \propto c_k \cdot v_j$$

- Normalise to turn this into a probability
- by passing through a softmax function:

$$p(w_k|w_j) = \frac{e^{c_k \cdot v_j}}{\sum_{i \in V} e^{c_i \cdot v_j}}$$

Skip-gram: Learning

- Start with some initial embeddings (usually random)
- At training time, walk through the corpus
- iteratively make the embeddings for each word
  - more like the embeddings of its neighbors
  - less like the embeddings of other words.

Skip-gram: Objective

Learn parameters C and W that maximize the overall corpus probability:

$$\arg\max \prod_{(w_j,w_k)\in D} p(w_k|w_j)$$

$$p(w_k|w_j) = \frac{e^{c_k \cdot v_j}}{\sum_{i\in V} e^{c_i \cdot v_j}}$$

$$\arg\max \sum_{(w_j,w_k)\in D} \log p(w_k|w_j) = \sum_{(w_j,w_k)\in D} (\log e^{c_k \cdot v_j} - \log \sum_{c_i \in V} e^{c_i \cdot v_j})$$

Skip-gram with negative sampling

Problem with softmax: expensive to compute the denominator for the whole vocabulary

$$p(w_k|w_j) = \frac{e^{c_k \cdot v_j}}{\sum_{i \in V} e^{c_i \cdot v_j}}$$

Approximate the denominator: **negative sampling**

- At training time, walk through the corpus
- for each target word and positive context
- sample k noise samples or negative samples, i.e. other words

Skip-gram with negative sampling

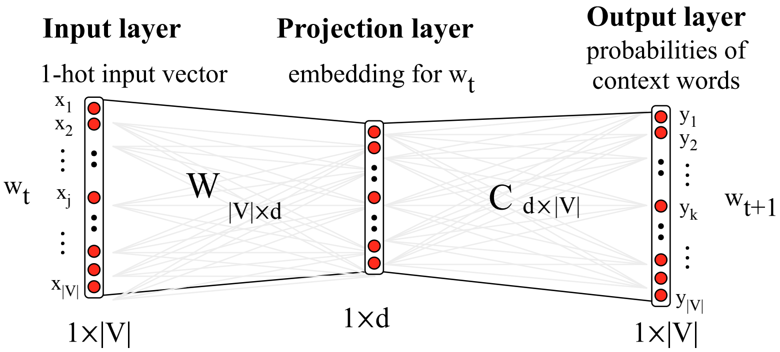
- Objective in training:
  - Make the word like the context words
    lemon, a [tablespoon of apricot preserves or] jam.

    c_1 c_2 w c_3 c_4
  - And not like the k negative examples

[cement idle dear coaxial apricot attendant whence forever puddle]

n_1 n_2 n_3 n_4 w n_5 n_6 n_7 n_8

Visualising skip-gram as a network



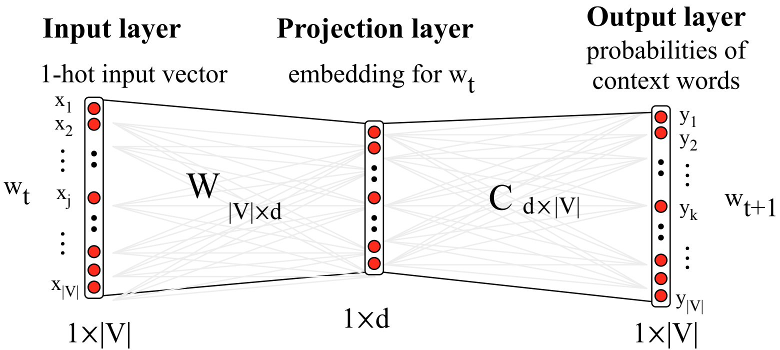
Slide credit: Dan Jurafsky

One hot vectors

- A vector of length |V|
- 1 for the target word and 0 for other words
- So if "bear" is vocabulary word 5
- The one-hot vector is [0,0,0,0,1,0,0,0,0........0]

$$\begin{array}{cccccccccccccccccccccc} w_0 & w_1 & & & & & & & & w_j & & & & & & & & & & w_{|V|} \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \end{array}$$

Visualising skip-gram as a network



Slide credit: Dan Jurafsky

Properties of embeddings

They capture similarity

FRANCE 454	JESUS 1973	XBOX 6909	REDDISH 11724	SCRATCHED 29869	MEGABITS 87025
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Slide credit: Ronan Collobert

Properties of embeddings

They capture analogy

Analogy task: ***a*** *is to* ***b*** *as* ***c*** *is to* ***d***
The system is given words a, b, c, and it needs to find d.

“apple” is to “apples” as “car”‘ is to ?
“man” is to “woman” as “king” is to ?

Solution: capture analogy via vector offsets

$$a - b \approx c - d$$

$$man - woman \approx king - queen$$

$$d_w = \operatorname*{argmax}_{d'_w \in V} cos(a - b, c - d')$$

Properties of embeddings

They capture analogy

Analogy task: ***a*** *is to* ***b*** *as* ***c*** *is to* ***d***
The system is given words a, b, c, and it needs to find d.

> *"apple" is to "apples" as "car"" is to ?*
> *"man" is to "woman" as "king" is to ?*

Solution: capture analogy via vector offsets

$$a - b \approx c - d$$

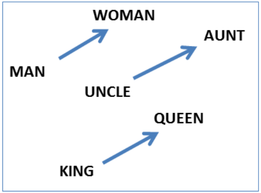
$$man - woman \approx king - queen$$

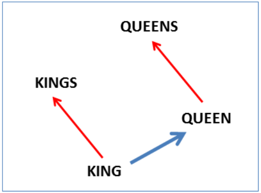
$$d_w = \operatorname*{argmax}_{d'_w \in V} cos(a - b, c - d')$$

Properties of embeddings

Capture analogy via vector offsets

$$man - woman \approx king - queen$$





Mikolov et al. 2013. *Linguistic Regularities in Continuous Space Word Representations*

Properties of embeddings

They capture a range of semantic relations

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Mikolov et al. 2013. *Efficient Estimation of Word Representations in Vector Space*

Word embeddings in practice

Word2vec is often used for pretraining in other tasks.

- It will help your models start from an **informed** position
- Requires only **plain text** - which we have a lot of
- Is very **fast** and easy to use
- Already **pretrained** vectors also available (trained on 100B words)

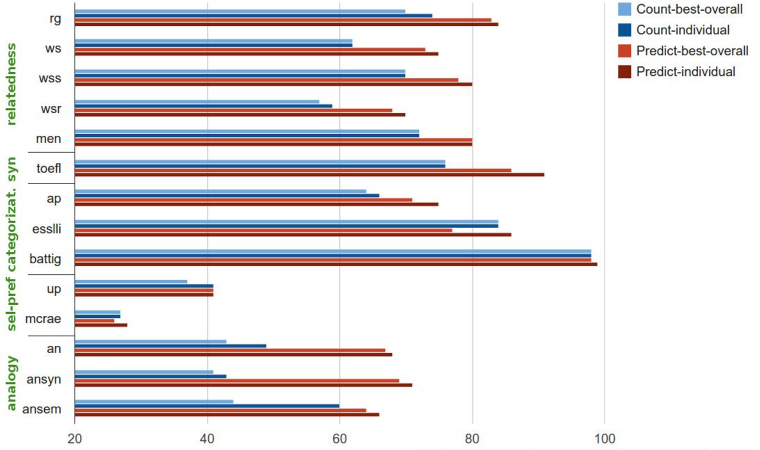
However, for best performance it is important to continue training, fine-tuning the embeddings for a specific task.

Count-based models vs. skip-gram word embeddings

Baroni et. al. 2014. *Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors.*

- Comparison of count-based and neural word vectors on 5 types of tasks and 14 different datasets:
  1. Semantic relatedness
  2. Synonym detection
  3. Concept categorization
  4. Selectional preferences
  5. Analogy recovery

Count-based models vs. skip-gram word embeddings



Some of these findings were later disputed by Levy et. al. 2015. *Improving Distributional Similarity with Lessons Learned from Word Embeddings*

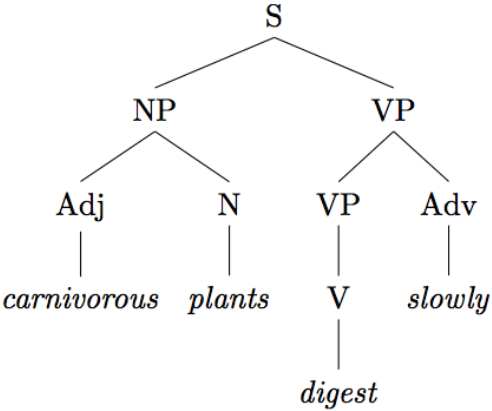
Outline.

- Word clustering (finishing off)
- Semantics with dense vectors
- Compositional semantics
- Compositional distributional semantics
- Semantic composition in recurrent neural networks

Compositional semantics

- Principle of Compositionality: meaning of each whole phrase derivable from meaning of its parts.
- Sentence structure conveys some meaning
- Deep grammars: model semantics alongside syntax, one semantic composition rule per syntax rule

Compositional semantics alongside syntax



Semantic composition is non-trivial

- Similar syntactic structures may have different meanings:

  it barks
  it rains; it snows – pleonastic pronouns

- Different syntactic structures may have the same meaning:

  Kim seems to sleep.
  It seems that Kim sleeps.

- Not all phrases are interpreted compositionally, e.g. idioms:

  red tape
  kick the bucket

  but they can be interpreted compositionally too, so we can not simply block them.

Semantic composition is non-trivial

- Elliptical constructions where additional meaning arises through composition, e.g. logical metonymy:

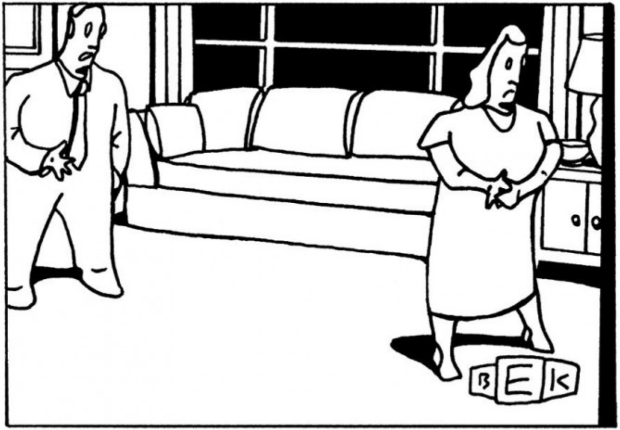
  fast programmer
  fast plane

- Meaning transfer and additional connotations that arise through composition, e.g. metaphor

  *I cant **buy** this story.*
  *This sum will **buy** you a ride on the train.*

- Recursion

Recursion



"Of course I care about how you imagined I thought you perceived I wanted you to feel."

Outline.

Word clustering (finishing off)

Semantics with dense vectors

Compositional semantics

Compositional distributional semantics

Semantic composition in recurrent neural networks

Compositional distributional semantics

Can distributional semantics be extended to account for the meaning of phrases and sentences?

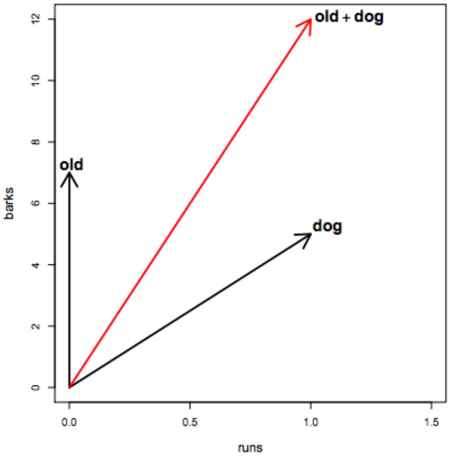
- Language can have an infinite number of sentences, given a limited vocabulary
- So we can not learn vectors for all phrases and sentences
- and need to do composition in a distributional space

1. Vector mixture models

Mitchell and Lapata, 2010. *Composition in Distributional Models of Semantics*

Models:

- Additive
- Multiplicative



Additive and multiplicative models

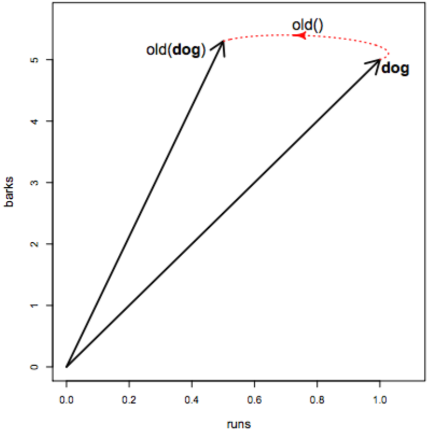
				additive		multiplicative	
	dog	**cat**	**old**	**old + dog**	**old + cat**	**old ⊙ dog**	**old ⊙ cat**
runs	1	4	0	1	4	0	0
barks	5	0	7	12	7	35	0

- correlate with human similarity judgments about adjective-noun, noun-noun, verb-noun and noun-verb pairs
- but... commutative, hence do not account for word order
  John hit the ball = *The ball hit John*!
- more suitable for modelling content words, would not port well to function words:
  e.g. *some dogs; lice and dogs; lice on dogs*

2. Lexical function models

Distinguish between:

- words whose meaning is directly determined by their distributional behaviour, e.g. nouns
- words that act as functions transforming the distributional profile of other words, e.g., verbs, adjectives and prepositions



Lexical function models

Baroni and Zamparelli, 2010. *Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space*

Adjectives as lexical functions

$$\textit{old dog} = \textit{old}(\textit{dog})$$

- Adjectives are parameter matrices ($\mathbf{A}_{old}$, $\mathbf{A}_{furry}$, etc.).
- Nouns are vectors (**house**, **dog**, etc.).
- Composition is simply **old dog** $= \mathbf{A}_{old} \times$ **dog**.

OLD	runs	barks
runs	0.5	0
barks	0.3	1

$\times$

	dog
runs	1
barks	5

$=$

	OLD(dog)
runs	$(0.5 \times 1) + (0 \times 5) = 0.5$
barks	$(0.3 \times 1) + (5 \times 1) = 5.3$

Learning adjective matrices

1. Obtain a distributional vector $\mathbf{n}_j$ for each noun n_j in the lexicon.
2. Collect adjective noun pairs (a_i, n_j) from the corpus.
3. Obtain a distributional vector $\mathbf{p}_{ij}$ of each pair (a_i, n_j) from the same corpus using a conventional DSM.
4. The set of tuples $\{(\mathbf{n}_j, \mathbf{p}_{ij})\}_j$ represents a dataset $\mathcal{D}(a_i)$ for the adjective a_i.
5. Learn matrix $\mathbf{A}_i$ from $\mathcal{D}(a_i)$ using linear regression.

Minimize the squared error loss:

$$L(\mathbf{A}_i) = \sum_{j \in \mathcal{D}(a_i)} \|\mathbf{p}_{ij} - \mathbf{A}_i \mathbf{n}_j\|^2$$

Polysemy in lexical function models

Generally:

- use single representation for all senses
- assume that ambiguity can be handled as long as contextual information is available

Exceptions:

- Kartsaklis and Sadrzadeh (2013): homonymy poses problems and is better handled with prior disambiguation
- Gutierrez et al (2016): literal and metaphorical senses better handled by separate models
- However, this is still an open research question.

Outline.

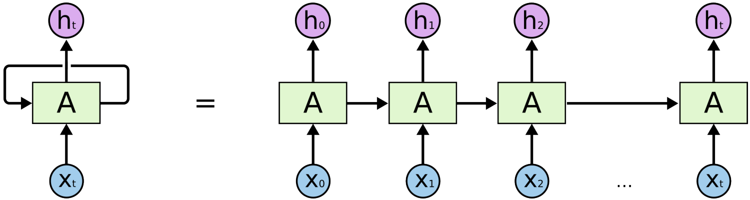
- Word clustering (finishing off)
- Semantics with dense vectors
- Compositional semantics
- Compositional distributional semantics
- Semantic composition in recurrent neural networks

Semantic composition in recurrent neural networks

An alternative is to perform semantic composition in recurrent neural networks (RNNs)

- Take word vectors as input
- Train phrase representations in a supervised setting, i.e. in a particular task
- Possible tasks: sentiment analysis; natural language inference; paraphrasing; machine translation etc.
- But any sequence labelling task would produce compositional representations when using RNNs.

Recurrent Neural Networks



Slide credit: Ann Copestake

Compositional representations in RNNs

- widely used in NLP today
- task-specific representations, i.e. not general purpose (as the ones learnt in an unsupervised way)
- oblivious of syntax

Acknowledgement

Today's lecture is partly based on materials of Dan Jurafsky and Marek Rei.