Word semantics

Paweł Rychlikowski

Instytut Informatyki UWr

30 kwietnia 2022

Hello World!

We start!

This lecture, and the next 5-6 will be about **natural language processing (NLP)**

Hello World!

We start!

This lecture, and the next 5-6 will be about **natural language processing (NLP)**

Our main goals:

- Show some flavour of NLP
- Describe the most influencial NN architectures in NLP
- Describe the most important NLP tasks

Hello World!

We start!

This lecture, and the next 5-6 will be about **natural language processing (NLP)**

Our main goals:

- Show some flavour of NLP
- Describe the most influencial NN architectures in NLP
- Describe the most important NLP tasks
- Prepare to Assigment 4 and 5, and to the final project

Bibliography

This part of the course is mainly based on:

- Natural Language Processing with Deep Learning (CS224N/Ling284), by Chris Manning
- Speech and Language Processing, 3rd edition draft, by D. Juraffsky, H. Martin (https://web.stanford.edu/~jurafsky/slp3/)
- Deep Learning book (Goodfellow, Bengio, Courville)

Bibliography

This part of the course is mainly based on:

- Natural Language Processing with Deep Learning (CS224N/Ling284), by Chris Manning
- Speech and Language Processing, 3rd edition draft, by D. Juraffsky, H. Martin (https://web.stanford.edu/~jurafsky/slp3/)
- Deep Learning book (Goodfellow, Bengio, Courville)

(some slides from CS224N will be just copied...)

Plans for first the lectures

1. Word semantics (aka. context free word embedings)
2. 3 simplest NN architectures for NLP (try to guess!)

Plans for first the lectures

1. Word semantics (aka. context free word embedings)
2. 3 simplest NN architectures for NLP (try to guess!)
3. Recurent Neural Networks (RNN, LSTM, GRU)

Plans for first the lectures

1. Word semantics (aka. context free word embedings)
2. 3 simplest NN architectures for NLP (try to guess!)
3. Recurent Neural Networks (RNN, LSTM, GRU)
4. Machine Translation and Attention

Plans for first the lectures

1. Word semantics (aka. context free word embedings)
2. 3 simplest NN architectures for NLP (try to guess!)
3. Recurent Neural Networks (RNN, LSTM, GRU)
4. Machine Translation and Attention
5. Transformers, BERT-like-family

Plans for first the lectures

1. Word semantics (aka. context free word embedings)
2. 3 simplest NN architectures for NLP (try to guess!)
3. Recurent Neural Networks (RNN, LSTM, GRU)
4. Machine Translation and Attention
5. Transformers, BERT-like-family
6. More on Natural Language Generation


ANNO
LITERARIA INSTRUXIT

Brief history of NLP

- Rule based systems
  - Noam Chomsky published his book, **Syntactic Structures**, in **1957**.
  - In **1964**, ELIZA, a "typewritten" comment and response process, designed to imitate a psychiatrist

Brief history of NLP

- Rule based systems
  - Noam Chomsky published his book, **Syntactic Structures**, in **1957**.
  - In **1964**, ELIZA, a "typewritten" comment and response process, designed to imitate a psychiatrist
- Using **corpora** to statistically model language
  - (for instance IBM automatic speech recognition, approx. **1980+**)

Brief history of NLP

- Rule based systems
  - Noam Chomsky published his book, **Syntactic Structures**, in **1957**.
  - In **1964**, ELIZA, a "typewritten" comment and response process, designed to imitate a psychiatrist
- Using **corpora** to statistically model language
  - (for instance IBM automatic speech recognition, approx. **1980+**)
- Machine Learning comes! (late 1980's)

Brief history of NLP

- Rule based systems
  - Noam Chomsky published his book, **Syntactic Structures**, in **1957**.
  - In **1964**, ELIZA, a "typewritten" comment and response process, designed to imitate a psychiatrist
- Using **corpora** to statistically model language
  - (for instance IBM automatic speech recognition, approx. **1980+**)
- Machine Learning comes! (late 1980's)
  - NBC, HMM, CRF, Logistic Regression, SVM, ...
  - Feature engineering (binary features like **w** in distance 2, and other)

Brief history of NLP

- Rule based systems
  - Noam Chomsky published his book, **Syntactic Structures**, in **1957**.
  - In **1964**, ELIZA, a "typewritten" comment and response process, designed to imitate a psychiatrist
- Using **corpora** to statistically model language
  - (for instance IBM automatic speech recognition, approx. **1980+**)
- Machine Learning comes! (late 1980's)
  - NBC, HMM, CRF, Logistic Regression, SVM, ...
  - Feature engineering (binary features like **w** in distance 2, and other)
- **Pretraining + Neural Networks and Deep Learning**

What is natural language?

What is natural language?

Answer

Language is a set of sequences!

What is natural language?

Answer

Language is a set of sequences!

Many options:

- Characters (ASCII, Extended ASCII, Unicode)
- Bytes (UTF-8)

What is natural language?

Answer

Language is a set of sequences!

Many options:

- Characters (ASCII, Extended ASCII, Unicode)
- Bytes (UTF-8)
- Words, Words++
- (sometimes common phrases, like New York, are treated as words)

What is natural language?

Answer

Language is a set of sequences!

Many options:

- Characters (ASCII, Extended ASCII, Unicode)
- Bytes (UTF-8)
- Words, Words++
- (sometimes common phrases, like New York, are treated as words)
- WordPieces (fixed number, approx. 30K, includes letters, and popular words)

Words++

Our first choice!

Words++

Our first choice!

Note: the number of **word types** can be huge!

Words++

Our first choice!

Note: the number of **word types** can be huge!

- You will need some technical *pseudowords*: `<out-of-vocab>`, `<begin-of-sentence>`, `<end-of-sentence>`, `<pad>`, ...

Words++

Our first choice!

Note: the number of **word types** can be huge!

- You will need some technical *pseudowords*: `<out-of-vocab>`, `<begin-of-sentence>`, `<end-of-sentence>`, `<pad>`, ...
- Other option: `<unknown-adj>`, `<unknown-noun>`, `<unknown-verb>`, `<unknown-other>`

Words++

Our first choice!

Note: the number of **word types** can be huge!

- You will need some technical *pseudowords*: `<out-of-vocab>`, `<begin-of-sentence>`, `<end-of-sentence>`, `<pad>`, ...
- Other option: `<unknown-adj>`, `<unknown-noun>`, `<unknown-verb>`, `<unknown-other>`
- Other option: `<-ology>`, `<-ator>`, `<-ization>`, ...

Words++

Our first choice!

Note: the number of **word types** can be huge!

- You will need some technical *pseudowords*: `<out-of-vocab>`, `<begin-of-sentence>`, `<end-of-sentence>`, `<pad>`, ...
- Other option: `<unknown-adj>`, `<unknown-noun>`, `<unknown-verb>`, `<unknown-other>`
- Other option: `<-ology>`, `<-ator>`, `<-ization>`, ...
- (you can just use last K letters for uncommon words)

Tokenization

Look for **tokenize** method in your favourite NLP/NN library!

Tokenization

Look for **tokenize** method in your favourite NLP/NN library!

Simple tokenization of s:

- For every punctuation character c, do
  `s = s.replace(c, ' ' + c + ' ')`
- Return `s.split()` or `s.lower().split()`

How do we have usable meaning in a computer?

Previously commonest NLP solution: Use, e.g., WordNet, a thesaurus containing lists of **synonym sets** and **hypernyms** ("is a" relationships)

e.g., synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
            ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g., hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

14

Problems with resources like WordNet

- A useful resource but missing nuance:
  - e.g., “proficient” is listed as a synonym for “good”
    This is only correct in some contexts
  - Also, WordNet list offensive synonyms in some synonym sets without any coverage of the connotations or appropriateness of words
- Missing new meanings of words:
  - e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
  - Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt
- Can’t be used to accurately compute word similarity (see following slides)

15

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:

hotel, conference, motel – a localist representation

Means one 1, the rest 0s

Such symbols for words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000+)

16

Problem with words as discrete symbols

Example: in web search, if a user searches for "Seattle motel", we would like to match documents containing "Seattle hotel"

But:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

These two vectors are orthogonal

There is no natural notion of **similarity** for one-hot vectors!

Solution:

- Could try to rely on WordNet's list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**

17

Representing words by their context

- Distributional semantics: **A word's meaning is given by the words that frequently appear close-by**
  - *"You shall know a word by the company it keeps"* (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- We use the many contexts of *w* to build up a representation of *w*

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

These context words will represent ***banking***

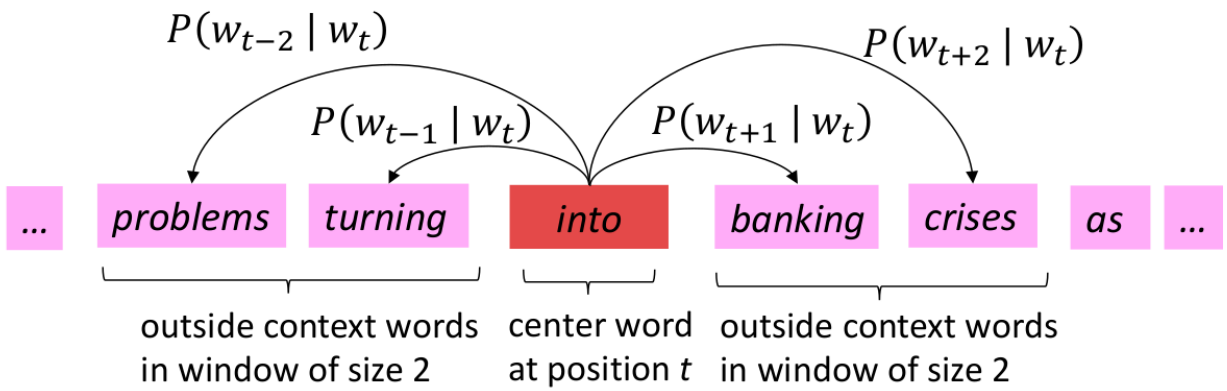
Word2Vec

The first robust dense vector representation of words!

- *Efficient Estimation of Word Representations in Vector Space* Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean
- *Distributed representations of words and phrases and their compositionality*, Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg S.; Dean, Jeff (2013).

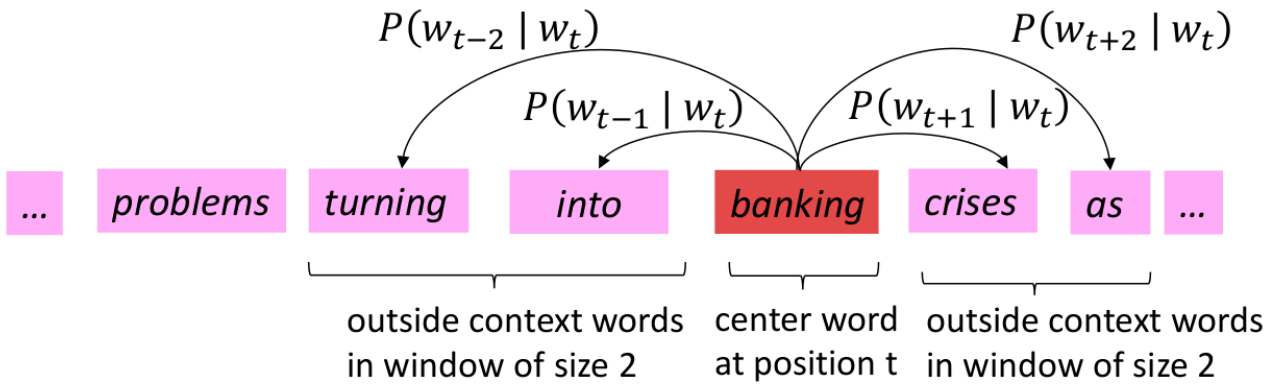
Word2Vec Overview

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2Vec Overview

Example windows and process for computing $P(w_{t+j} \mid w_t)$



W2V versions

1. Skip-grams: $P(w_{t+j}|w_t)$

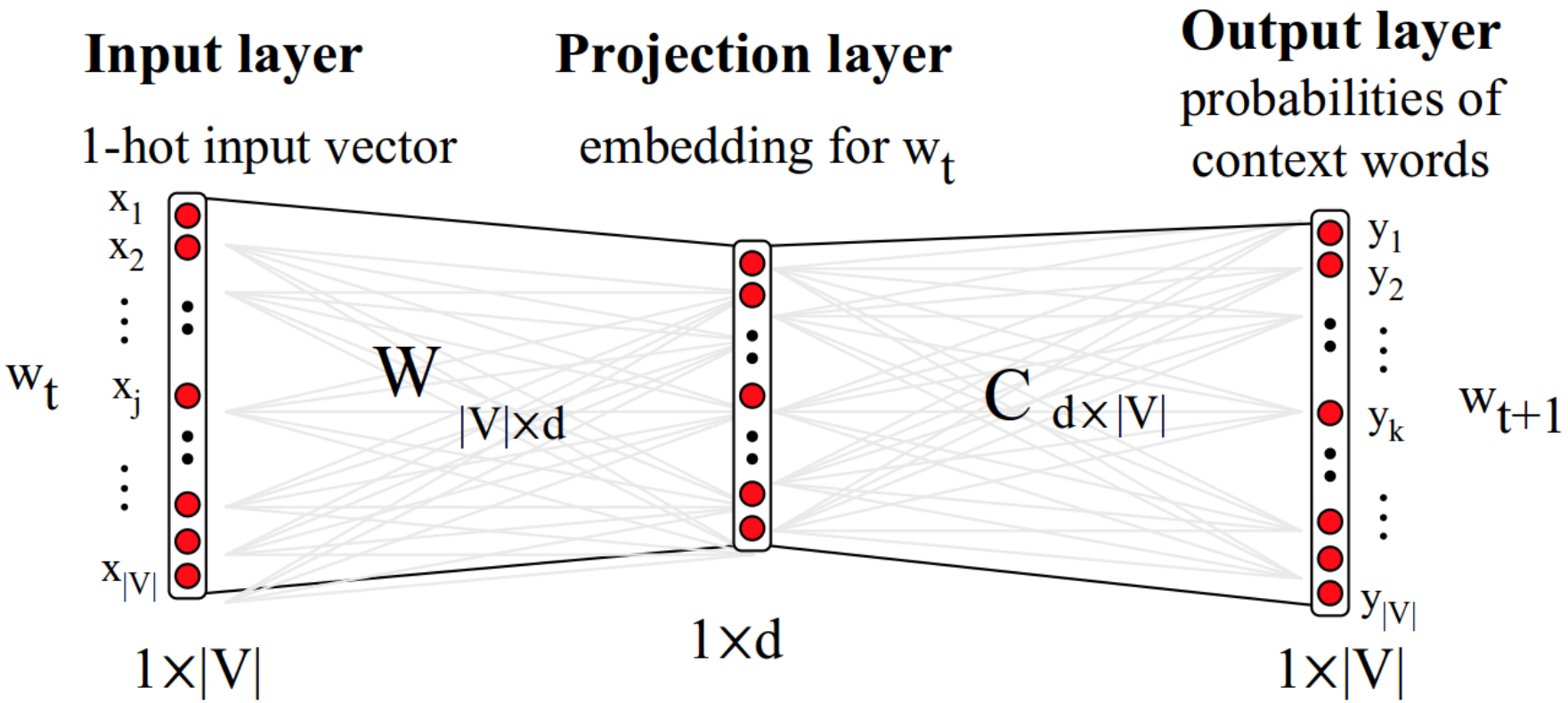
W2V versions

1. Skip-grams: $P(w_{t+j}|w_t)$
2. Bigrams: $P(w_{t+1}|w_t)$

W2V versions

1. Skip-grams: $P(w_{t+j}|w_t)$
2. Bigrams: $P(w_{t+1}|w_t)$
3. **CBOW** (continuous bag of words)

Word2Vec with skip-grams as Neural Network



Word2vec: objective function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_t. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

24

Word2vec: prediction function

② Exponentiation makes anything positive

① Dot product compares similarity of *o* and *c*.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

③ Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \to (0,1)^n$ ← Open region

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in Deep Learning

But sort of a weird name because it returns a distribution!

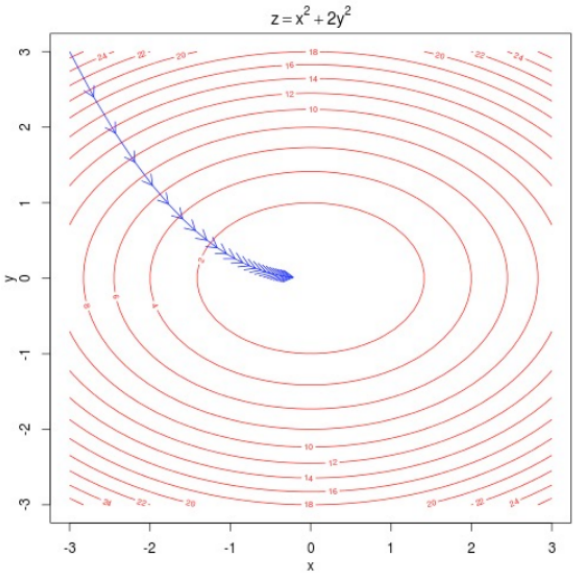
27

To train the model: Optimize value of parameters to minimize loss

To train a model, we gradually adjust parameters to minimize a loss

- Recall: θ represents **all** the model parameters, in one long vector
- In our case, with *d*-dimensional vectors and *V*-many words, we have →
- Remember: every word has two vectors

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



- We optimize these parameters by walking down the gradient (see right figure)
- We compute all vector gradients!

28

Say 'hello' to aardvark!

4. Objective Function

Maximize $J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} p(w'_{t+j} | w_t; \theta)$

Or minimize neg. log likelihood ave.

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log p(w'_{t+j} | w_t)$$

[negate to minimize; log is monotone]

text length

window size

where

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

word IDs

We now take derivatives to work out minimum

Each word type (vocab entry) has two word representations: as center word and context word

29

$$\frac{\partial}{\partial v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w v_c)}$$

$$= \underbrace{\frac{\partial}{\partial v_c} \log \exp(u_o^T v_c)}_{①} - \underbrace{\frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp(u_w^T v_c)}_{②}$$

① $\frac{\partial}{\partial v_c} \log \exp(u_o^T v_c) = \frac{\partial}{\partial v_c} u_o^T v_c = u_o$

$\frac{\partial}{\partial v_c}$: Vector! Not high school single variable calculus

$\log \exp$: inverses

You can do things one variable at a time, and this may be helpful when things get gnarly.

$$\forall j \quad \frac{\partial}{\partial (v_c)_j} u_o^T v_c = \frac{\partial}{\partial (v_c)_j} \sum_{i=1}^{d} (u_o)_i (v_c)_i$$

$$= (u_o)_j$$

Each term is zero except when $i = j$

$$\textcircled{2}\ \frac{\partial}{\partial v_c} \underbrace{\log}_{f} \underbrace{\sum_{w=1}^{V} \exp(u_w^T v_c)}_{z = g(v_c)}$$

$$= \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot \frac{\partial}{\partial v_c} \sum_{x=1}^{V} \exp(u_x^T v_c)$$

Important to change index

$$\frac{\partial}{\partial v_c} f(\overbrace{g(v_c)}^{z}) = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial v_c}$$

Use chain rule

$$= \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot \left(\sum_{x=1}^{V} \frac{\partial}{\partial v_c} \underbrace{\exp}_{f} \underbrace{(u_x^T v_c)}_{z = g(v_c)} \right)$$

Move deriv inside sum

$$\left(\sum_{x=1}^{V} \exp(u_x^T v_c) \frac{\partial}{\partial v_c} u_x^T v_c \right)$$

Chain rule

$$\left(\sum_{x=1}^{V} \exp(u_x^T v_c)\, u_x \right)$$

31

$$\frac{\partial}{\partial v_c} \log\left(p(o|c)\right) = u_o - \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot \left(\sum_{x=1}^{V} \exp(u_x^T v_c)\, u_x\right)$$

$$= u_o - \sum_{x=1}^{V} \frac{\exp(u_x^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}\, u_x$$

Distribute term across sum

$$= u_o - \sum_{x=1}^{V} p(x|c)\, u_x$$

This an expectation: average over all context vectors weighted by their probability

$$= \text{observed} - \text{expected}$$

This is just the derivatives for the center vector parameters
Also need derivatives for output vector parameters (they're similar)
Then we have derivative w.r.t. all parameters and can minimize

32

Problem with large Softmax

Large sum in the gradient!

Problem with large Softmax

Large sum in the gradient!

Solution: hierarchical softmax (decompose the decision to the list of decisions)
(probably) Better solution: negative sampling

Negative sampling

- We can easily obtain related objects
  - Two consecutive words, sentences, or sentences in one article, ...

Negative sampling

- We can easily obtain related objects
  - Two consecutive words, sentences, or sentences in one article, ...
- We can **sample** unrelated objects

Intuition of one step of gradient descent