Word representations

M. Pikuliak // NN Group // 17. 10. 2018

Outline

1. Text representation
2. Pre-trained word embeddings
3. Advanced topics

Text representation

How to represent our text data so the neural networks understand them

How can we represent text?

- **Words**
  - Each word is an indivisible symbol.
  - Text is a sequence of such symbols.

How can we represent text?

- **Words**
- **Sub-words**
  - With words alone we lose morphological information.
  - Sub-words (characters or N-gram of characters) can be symbols instead.
  - Not suitable for logographic languages.

How can we represent text?

- **Words**
- **Sub-words**
- **Feature engineering**
  - Hand-crafted features extracted from the text.
  - E.g. number of elongated words (cool > cooool) is a good predictor of strong sentiment.

Word level representation

- *Word is the smallest element that can be uttered in isolation with objective or practical meaning.* [Wikipedia]
- Words are symbols, we do not care about their *form.*
- We need numerical representation for this phenomenon.
- **Idea:** Assign a number to each word and use this number as the word representation.

Vocabulary building

What is our task

Input: Raw data corpus

Lorem ipsum dolor sit amet. Ipsum consectetur adipiscing elit.

Output: Vocabulary

```
1: lorem
2: ipsum
3: dolor
4: sit
5: amet
6: .
7: consectetur
8: adipiscing
9: elit
```

1. Tokenization

- **NLP task:** split text into individual tokens.
- `str.split` is not enough – punctuation, *scriptio continua*

I met my co-worker.

I	*met*	*my*	*co-worker.*			
I	*met*	*my*	*co*	*-*	*worker*	.
I	*met*	*my*	*co-worker*	.		

1. Tokenization

- **NLP task:** split text into individual tokens.
- `str.split` is not enough – punctuation, *scriptio continua*

don’t

do n’t

do not

don ‘ t

1. Tokenization

- **NLP task:** split text into individual tokens.
- `str.split` is not enough – punctuation, *scriptio continua*

- If possible use off-the-shelf tokenizer.
- `nltk punkt` available for 17 languages.

2. Word post-processing

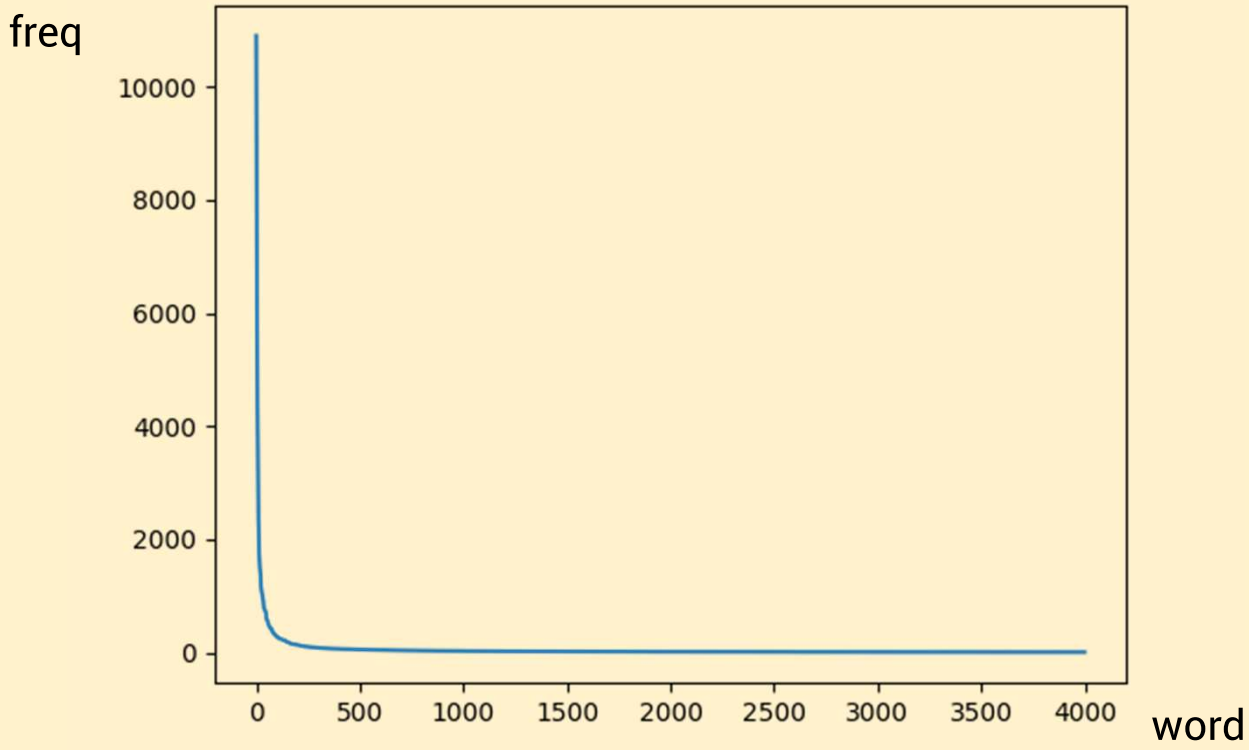
1. Letter case
2. Common token for numbers @NUMBER
3. Common token for URLs, email addresses, etc. @URL
4. Common token for emojis with the same emotion

We met 5 days ago :-)

we met @NUMBER day ago ☺

3. Word filtering

- **Zipf's law** can be observed on word frequencies.



3. Word filtering

- **Zipf's law** can be observed on word frequencies.
- We often limit the number of words we have in our vocabulary to fixed number or frequency.
- Model can't learn much about rare words – overfitting.
- More words = more memory.

3. Word filtering

1: .	150x	1: .
2: ipsum	70x	2: ipsum
3: lorem	50x	3: lorem
4: sit	40x	4: sit
5: amet	30x	5: amet
6: dolor	20x	6: @OOV
7: consectetur	10x	
8: adipiscing	7x	OOV = out of vocabulary
9: elit	5x	

4. Final word representations

Vocabulary

```
1: .
2: ipsum
3: lorem
4: sit
5: amet
6: @OOV
```

Text processing

Lorem ipsum dolor sit amet. Ipsum consectetur adipiscing elit.

```
lorem ipsum @OOV sit amet .
ipsum @OOV @OOV @OOV .
```

```
3 2 6 4 5 1 2 6 6 6 1
```

One-hot encoding

- Formally words are categorical data now.
- One-hot encoding is proper representation in such cases.
- For word n we have vector with size V where n-th unit is one.

	1	2	3		$V-1$	V
word 2	$[0,$	$1,$	$0,$	$\dots,$	$0,$	$0]$
word $V-1$	$[0,$	$0,$	$0,$	$\dots,$	$1,$	$0]$

Bag of words

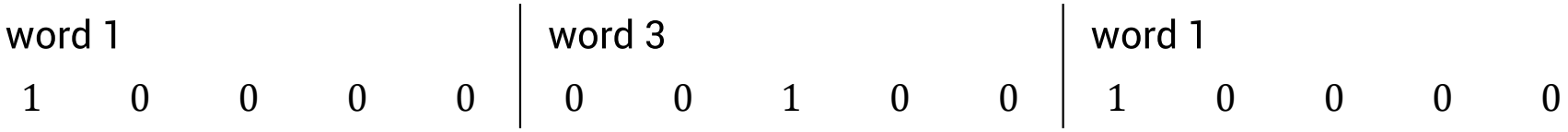
- Sum of word representations.
- For sequence $1\ \ 3\ \ 1$ with $V = 5$:

word 1	1	0	0	0	0
word 3	0	0	1	0	0
word 1	1	0	0	0	0
BoW	2	0	1	0	0

- This is *TF (term frequency)* representation.
- Binary and TF-IDF representations are related.

Fixed length sequence

- Concatenation of word representations
- For sequence `1 3 1` with $V = 5$:



- Output is $V \times K$, with K being the fixed length, here $K = 3$

Sequence

- List of word representations
- For sequence $1\ \ 3\ \ 1$ with $V = 5$:

word 1	1	0	0	0	0
word 3	0	0	1	0	0
word 1	1	0	0	0	0

	BoW	Fixed	Sequence
Preserve word order	No	Yes	Yes
Unlimited text length	Yes	No	Yes
Fixed representation length	Yes	Yes	No
Computation time increases with longer sentence	No	No	Yes

What representation should you use?

Fixed

- never

BoW

- when you don't care about word order (e.g. document classification)
- when you don't have much data
- when you don't care about performance hit if you can have smaller, simpler, quicker model

Sequence

- otherwise

Embedding layer

Common first operation with one-hot representation $\boldsymbol{x}$ is called *embedding*:

$$\boldsymbol{e} = \boldsymbol{W}\boldsymbol{x}$$

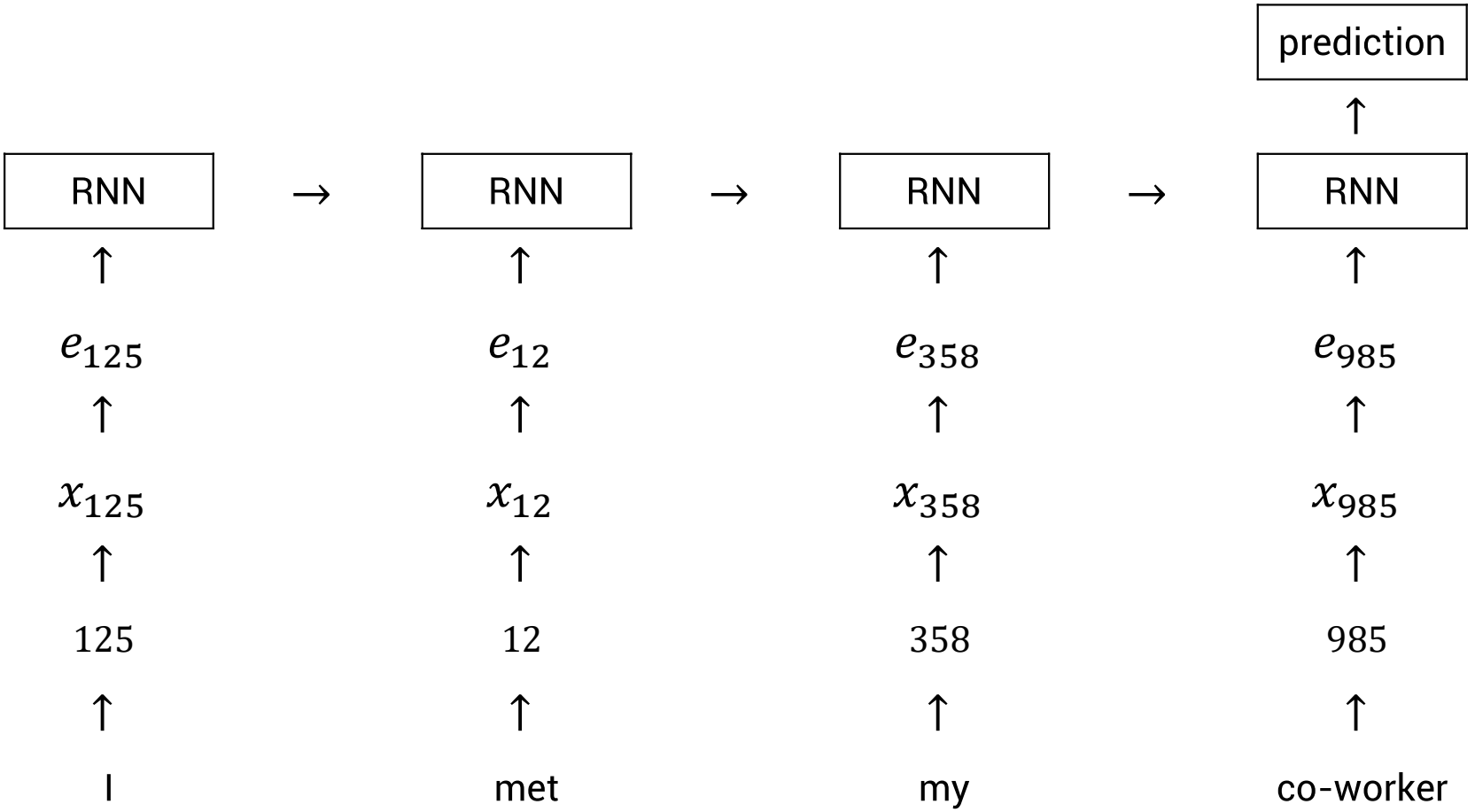
$\boldsymbol{e}$ is in fact n-th column of embedding matrix $\boldsymbol{W}$ for word n.

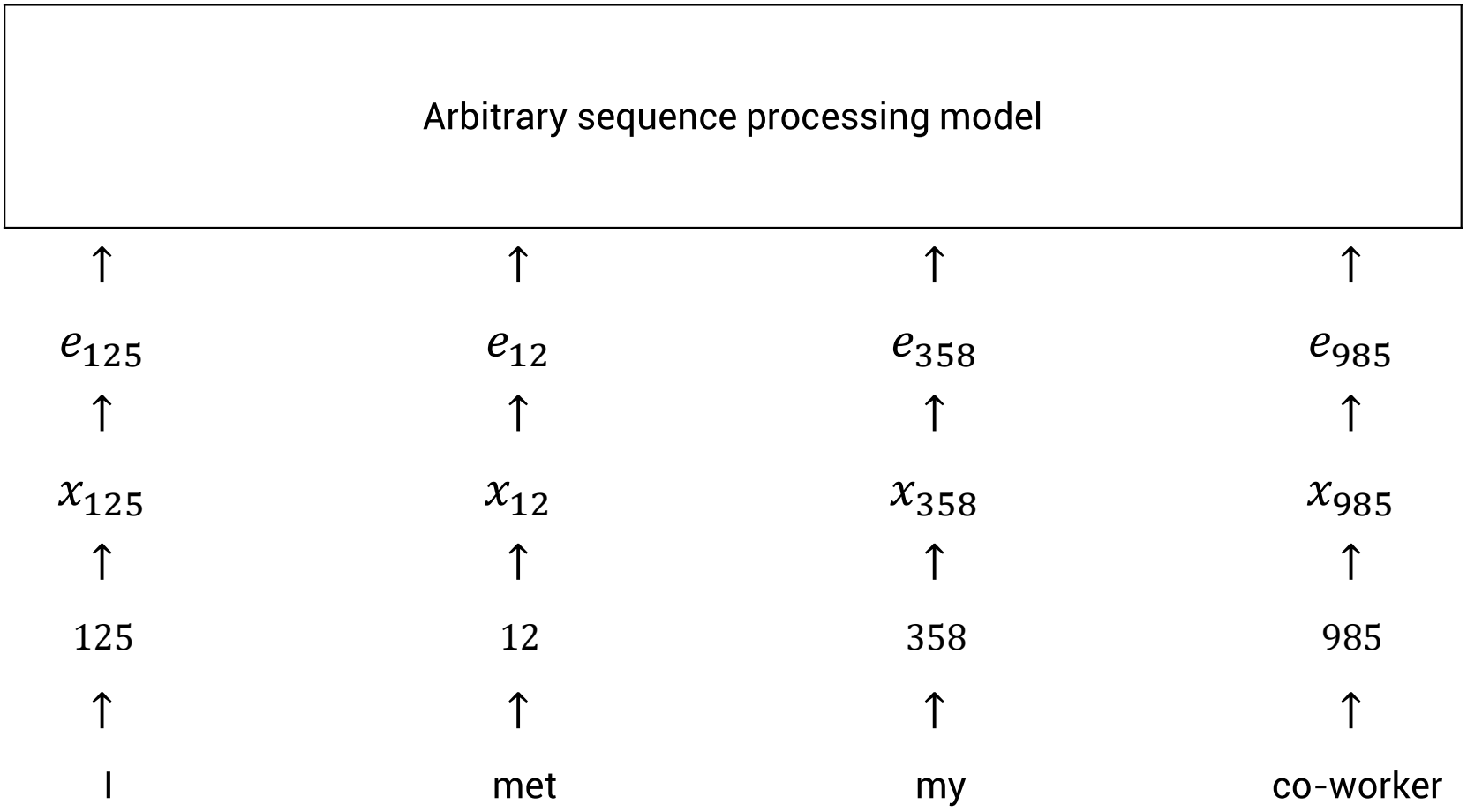
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \end{bmatrix}$$

Embedding layer

- $\boldsymbol{e}$ is also called the embedding of word n.
- It is the model's internal representation for this particular word.
- $\boldsymbol{W}$ has size $h \times V$, h is arbitrarily set (hundreds).

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

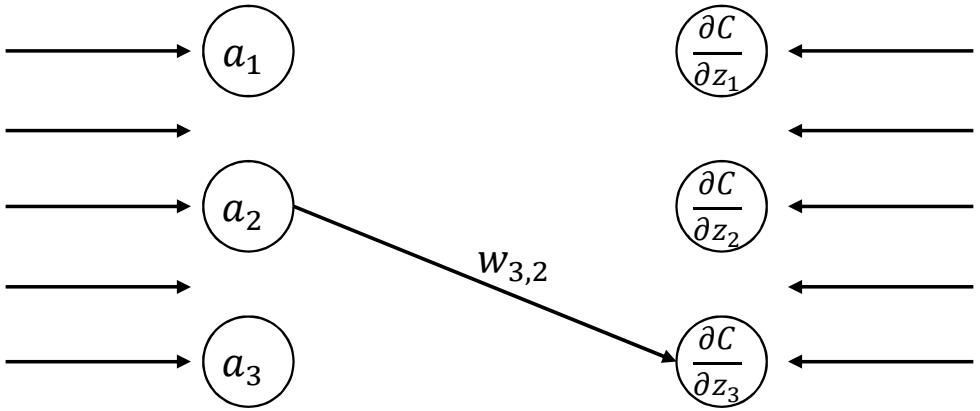

prediction
RNN
RNN
RNN
RNN
e_{125}
e_{12}
e_{358}
e_{985}
x_{125}
x_{12}
x_{358}
x_{985}
125
12
358
985
I
met
my
co-worker


Arbitrary sequence processing model
e_{125}
e_{12}
e_{358}
e_{985}
x_{125}
x_{12}
x_{358}
x_{985}
125
12
358
985
I
met
my
co-worker

Embedding learning

During back-propagation we calculate how much do individual weights contribute to error and change them accordingly:

$$\frac{\partial C}{\partial w_{j,k}} = a_k \frac{\partial C}{\partial z_j}$$

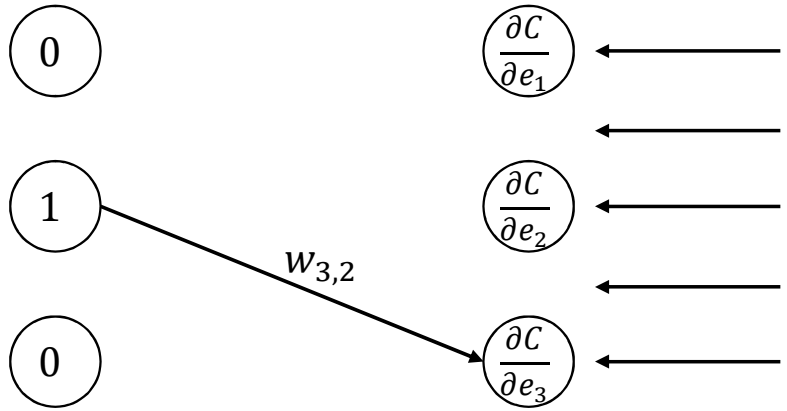


Embedding learning

During back-propagation we calculate how much do individual weights contribute to error and change them accordingly:

$$\frac{\partial C}{\partial w_{j,k}} = x_k \frac{\partial C}{\partial e_j}$$

$$\frac{\partial C}{\partial w_{j,k}} = \begin{cases} \frac{\partial C}{\partial e_j} & if\ k = n \\ 0 & otherwise \end{cases}$$



Why are we calculating the word embeddings all over again?

We should reuse trained embeddings for other tasks.
[Collobert & Weston, 2008]

Benefits:

- Faster training
- More robust word representations
- Better performance

Pre-trained word embeddings

word2vec, GloVe, fastText

What is a good auxiliary task?

- We need something general that is able to capture various types of information – semantic, syntactic.
- We need a task with an abundance of data.
- We need something we are able to train. Bonus points if we are able to do it quickly.

Language Modeling

Language Modeling

- Traditional linguistic task – First attempts in early 20th century.
- We want to model the probabilities of words following each other.

$w_1 w_2 w_3 w_4 w_5 \ldots ?$

- We calculate probability for each word in our vocabulary.
- Basically guessing the next word.

Language Modeling

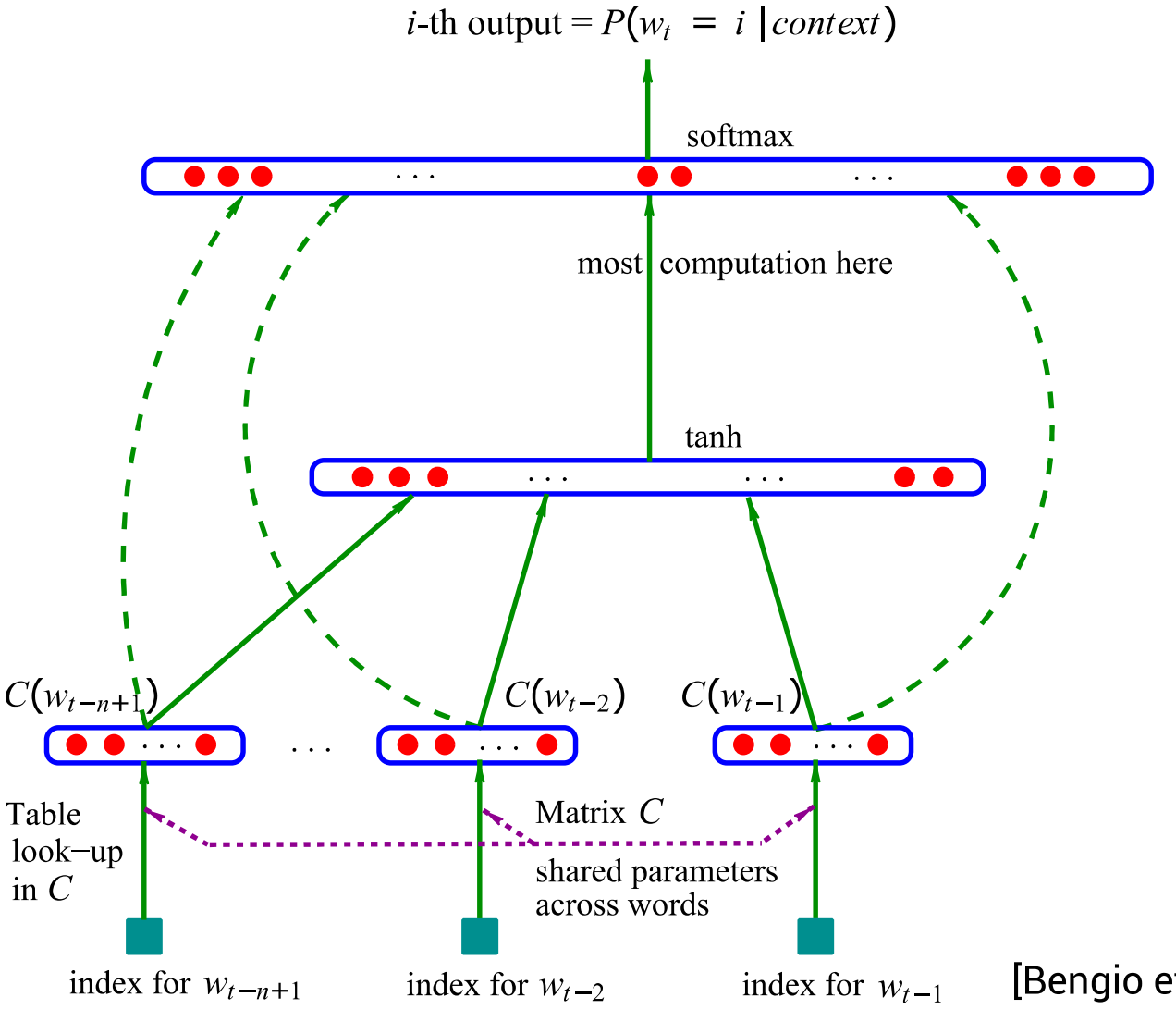
We use context to predict the missing word(s).

$w_1 w_2 w_3 w_4 w_5 w_6$

$w_1 w_2 w_3 w_4 w_5 w_6$

$w_1 w_2 w_3 w_4 w_5 w_6$

$w_1 w_2 w_3 w_4 w_5 w_6$



[Bengio et al, 2003]

word2vec

word2vec

- Open source library published by Tomáš Mikolov in 2013.
- Two conference papers published alongside [Mikolov et al, 2013a, 2013b].
- *Single use case*: compute word embeddings.

Why it got so popular?

1. Really fast training (hours instead of weeks).
2. Good results (used in practice to these days).

Task definition

- Take arbitrary corpus and *for each word* check its context:

  a b c d e f g a b c d e f

- Create training samples (w_t, w_c - training word, context word):

  g d, g e, g f, g a, g b, g c

- Train a model that is able to predict w_c from w_t

Architecture

$$\boldsymbol{e} = \boldsymbol{U}\boldsymbol{x}$$

$$\boldsymbol{y} = softmax(\boldsymbol{V}\boldsymbol{e})$$

$\boldsymbol{x}$		$\boldsymbol{e}$		$\boldsymbol{y}'$		$\boldsymbol{y}$
0				2.5		0.67
1		0.28		0.25		0.07
0	$\boldsymbol{U}$	0.52	$\boldsymbol{V}$	−2.5	$smax$	0.01
0		−0.32		1.2		0.18
0				0.25		0.07

Probabilities for each word appearing next to word 2.

Training

During training we want to maximize the probability for sample context words:

$$\frac{1}{N}\sum P(w_c|w_t)$$

But calculating the softmax is really expensive ☹

Training efficiency

- **Negative sampling**
  - Instead of calculating the probability for each word, we calculate it for w_c and a handful of other words
  - We are happy if w_c has higher probability than k random words

$$\log \sigma(y'_c) + \sum_{i=1}^{k} \boldsymbol{E}_{w_i \sim P_n(w)} [\log \sigma(-y'_i)]$$

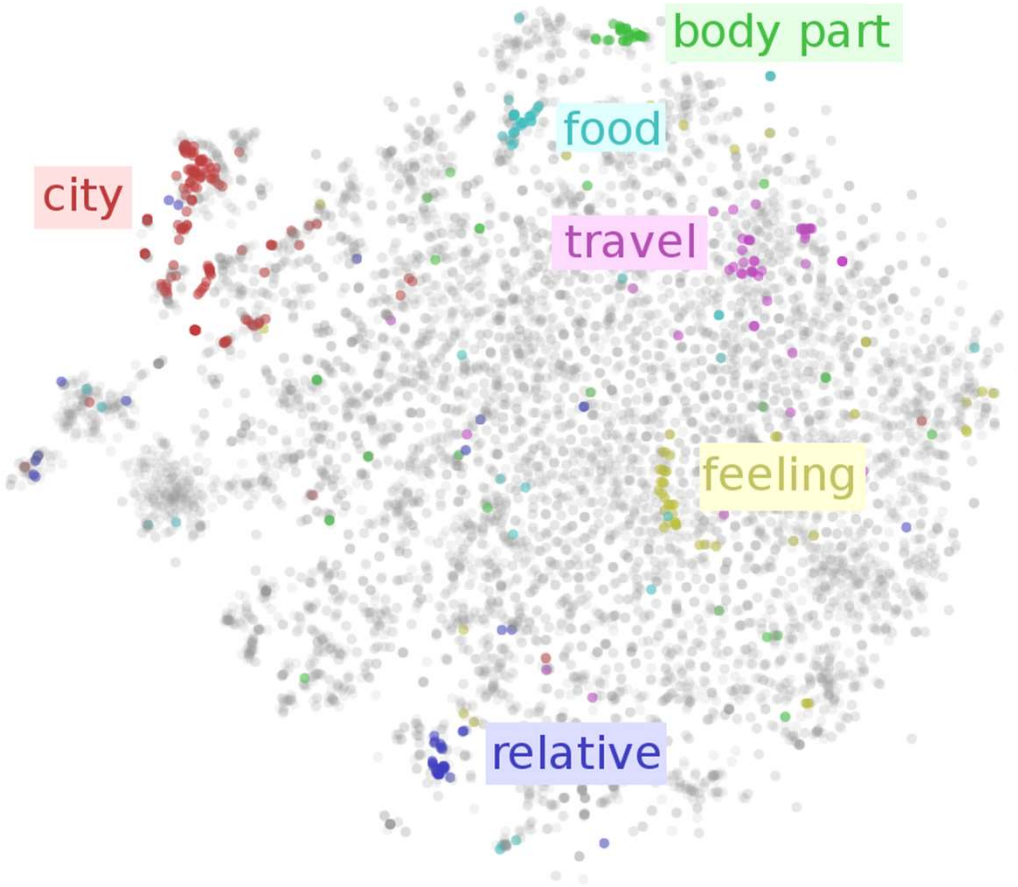
- **Frequent words subsampling**
  - We can make it even faster if we don't calculate this for frequent words millions of times.

After training

- We can predict w_c really good, but this is in practice not useful at all.
- *But,* we have a matrix $\boldsymbol{U}$ trained and every column there is a really good word representation for one word.
- In fact, similar representations are also in matrix $\boldsymbol{V}$ – we can use whatever, or their average.

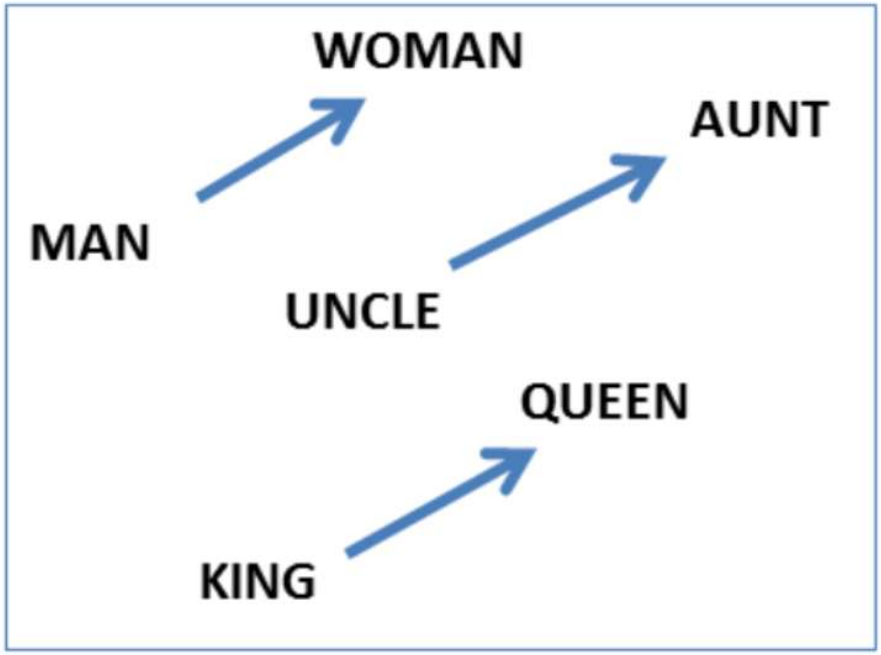
airport -0.0211522 -0.0639906 -0.0156481 -0.0515685 -0.085993 -0.0306212 0.024145 0.0724278 -0.0652632 -0.0835557 0.0362597 0.0692661 -0.0524079 0.0178493 -0.0863435 -0.00681157 -0.00132345 -0.0108425 0.0130419 0.0978561 -0.0228005 0.0280041 -0.0948472 -0.000208736 0.0542125 -0.0635952 0.137828 0.0127478 0.0682092 0.0387402 -0.0395706 0.0857594 -0.0121018 0.0564971 0.0761035 0.0207982 0.0352783 -0.0206669 -0.0631835 -0.0399481 0.0560315 -0.0748921 0.0424681 -0.0443015 0.112608 0.0145023 0.0872422 9.75901e-06 0.0610608 0.0202733 0.0901811 0.085763 0.0446861 0.0593011 -0.00223116 0.00147196 0.059941 0.0527836 -0.0123346 0.0497818 0.0416412 -0.0697532 0.0345629 0.0101009 -0.0185837 0.0153257 -0.0340884 -0.00372249 -0.0264511 0.0212026 -0.00851267 0.0452235 0.0680834 -0.0223026 0.0349835 -0.00828439 0.0997812 0.00793101 0.0152292 0.0416233 -0.125991 0.0647545 -0.0638486 -0.0518273 0.0912182 -0.0480815 -0.0235105 0.0464763 0.106834 -0.0940635 0.00289945 0.0136183 0.0122537 -0.024677 0.0581363 0.0914662 -0.0513043 0.0714464 -0.0310993 -0.0609727 0.0128154 0.0385586 -0.0285253 -0.100513 -0.00862626 -0.0236579 -0.0184705 -0.0166202 0.0544516 0.0765637 -0.00981527 -0.07633 -0.0509879 -0.0432877 -0.0178822 -0.0485398 0.0263648 0.055016 0.0160125 -0.0227286 0.222649 0.029203 -0.107942 -0.056941 0.0750161 -0.0361627 0.0278423 0.0239203 0.122521 -0.0591123 0.0440175 -0.00911229 -0.00283007 0.0319315 0.00030143 -0.000992797 0.00492049 -0.0827774 -0.00840248 0.0521527 -0.106205 -0.0113122 0.0471108 0.0209348 0.02978 0.0276015 0.192685 -0.0887701 -0.00634837 0.0359344 0.0792005 0.0264098 -0.0630487 -0.0208557 0.00492893 -0.0718148 0.0560567 -0.0718993 0.00700551 0.00194555 -0.0212457 0.0144701 -0.107302 0.0843681 0.0494817 0.0733822 -0.132931 -0.0961719 -0.0633183 0.0354491 0.0264978 -0.0522012 0.000745919 -0.0457484 -0.118879 0.0846773 -0.00357205 0.00885562 0.0147579 -0.0689911 -0.0385245 -0.0393208 0.0401098 0.0296092 0.0642602 0.00400236 -0.124234 -0.0680564 0.0326001 0.0227232 0.137073 -0.0567865 -0.0432553 0.0175058 -0.0382333 -0.0171806 -0.0392417 -0.0605826 -0.0420241 0.0524205 -0.0952229 0.0467406 0.0017758 -0.0580303 0.0538207 -0.0847798 -0.025006 0.00558805 0.068344 0.0528842 0.0285092 -0.116706 0.00345216 -0.019371 -0.054892 0.014008 -0.0659534 -0.119959 -0.00465662 -0.0169757 0.055131 0.053614 -0.143173 -0.00921439 0.144703 -0.0822202 0.0748633 -0.051644 -0.0667694 0.0332688 0.0541568 0.00382333 -0.0171294 0.0399786 -0.0672745 0.0322856 0.0104133 0.033134 0.0112928 0.0718795 0.00295445 0.0171172 -0.0990262 -0.00680204 -0.0800093 0.0442224 -0.0132602 -0.00919821 -0.034234 -0.0570111 0.0879702 -0.0527063 -0.0680294 0.00519172 0.00617528 -0.0652668 0.0228418 0.0066667 0.052629 -0.0160211 0.0310346 0.0129103 -0.0471055 0.00957028 -0.0383358 0.0182781 -0.00456837 0.0371207 0.00556972 -0.0449144 -0.0266021 -0.0711965 -0.0241719 -0.0831944 0.0427053 -0.14991 0.093287 0.0732636 0.0593855 -0.0115703 -0.0206939 -0.0460935 -0.0361339 -0.0373472 -0.0201367 -0.0282252 -0.00334377 -0.0325678 -0.0344928 0.0017051 0.0831692 -0.0317859 -0.00961845 0.0234673 -0.0500658 0.0114953 -0.0371441 0.102201 0.0277884 -0.0194411

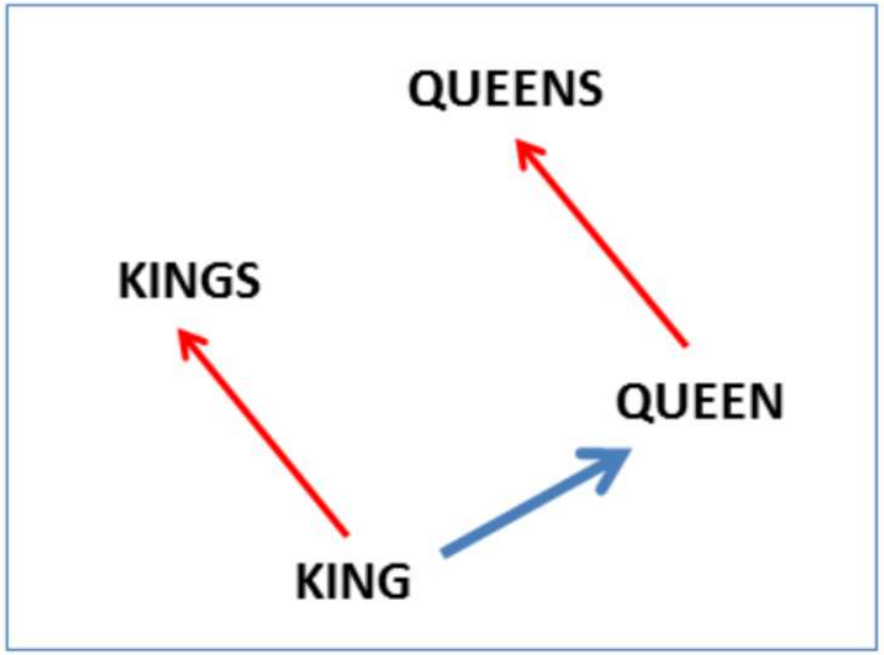
Similar words are clustered



[Olah, 2015]

Differences are semantically meaningful





[Mikolov et al, 2013c]

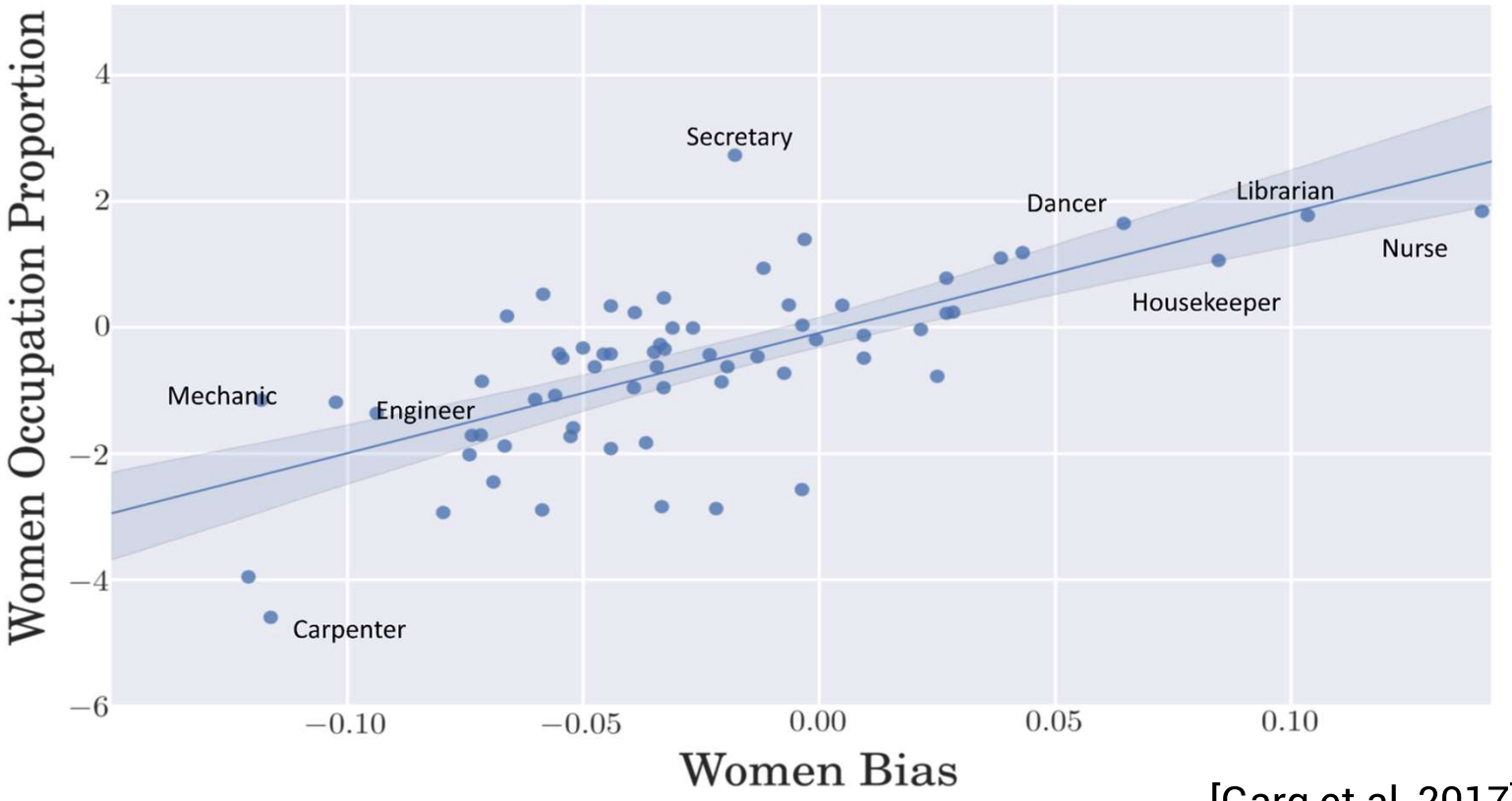
word2vec demo

http://bionlp-www.utu.fi/wv_demo/

Historical linguistics

[Hamilton et al, 2016]

Sociolinguistics



[Garg et al, 2017]

Averaging embeddings for documents

- We can create sentence / paragraph / document representations with pooling operations.
- Equal to using $\boldsymbol{W}$ on BoW representation.
- *Deep averaging networks* [Iyyer et al, 2015]

Embedding properties

- They are trained using a very rudimentary training objective.
- Yet they seemingly capture a lot of semantic information about words.

- **Back to the start:** They are used to initialize embedding matrices in NLP models and empirically they significantly improve results.
- **New insight:** They can be used as off-the-shelf word representations whenever we need.

Why does it work so well?

- $\boldsymbol{VU}$ is approximating co-occurrence matrix.
- Embeddings are “compressions" of columns / rows
- **Intuitive interpretation:** words that are used in similar context are semantically similar (cat, dog – fur, pet, etc.)
- *Harris' distributional hypothesis*

	A	**B**	**C**	**D**	**E**
A	0	0.32	0.21	0.07	0.4
B	0.22	0.01	0.35	0.22	0.2
C	0.13	0.13	0	0.52	0.22
D	0.54	0.32	0.08	0.01	0.05
E	0.1	0.2	0.52	0.18	0

GloVe

[Pennington et al, 2014]

- Quite popular alternative to word2vec.
- Conceptually very similar, slightly different math.

fastText

[Bojanowski et al, 2017]

- Each word is an indivisible symbol?
- *krásny, krásna, krajší, najkrajším, ...*
- *fastText* is a *word2vec* extension working with sub-words.

fastText

- *krásny - <krásny>, <kr, krá, rás, ásn, sny, ny>*
- We add all N-gram to our vocabulary.
- Input word representation is then *multi-hot.*

	<abc>	**<bca>**	**<a**	**ab**	**bc**	**c>**	**<b**	**ca**	**a>**
abc	1	0	1	1	1	1	0	0	0
bca	0	1	0	0	1	0	1	1	1

fastText

- *krásny - <krásny>, <kr, krá, rás, ásn, sny, ny>*
- We add all N-gram to our vocabulary.
- Input word representation is then *multi-hot.*
- There is an interaction between words now.

	<abc>	**<bca>**	**<a**	**ab**	**bc**	**c>**	**<b**	**ca**	**a>**
abc	1	0	1	1	1	1	0	0	0
bca	0	1	0	0	1	0	1	1	1
bc	0	0	0	0	1	1	1	0	0

And many others…

- Multilingual
- Sparse
- Compressed
- Paragraph embeddings
- Word sense sensitive
- Task specific embeddings
- Embeddings trained on non-LM tasks

…

What embedding model to use?

- *fastText* (available for 157 languages).
- *word2vec* is okay for morphologically simple languages.

How to use them in your model?

```
emb = tf.nn.embedding_lookup(pretrained_emb, word_ids)
```

Use existing or train your own?

- If you don't care about performance that much, *use existing*.
- If you care about performance, but you don't have much data, *use existing*.
- If you care about performance, and you have data, *train your own*.
- If you have **a lot** of data you might not even need pre-trained embeddings.

Should the embedding layer be fixed during training?

- Embedding matrix W is a weight matrix like any other.
- If we don't have much data – *keep it fixed*.
- Otherwise – *you can train it.*

- If we train it with small dataset we can observe a *semantic shift.*
- Words that are not in training set don't move while other words do.

How to deal with OOVs?

- Pre-trained embeddings restrict vocabulary.
  - Always check the words that are not found in embeddings.
- If you train your W you can randomly initialize new embeddings words that don't have embeddings.
- @OOV token is assigned its own embedding (zero vector, random vector, token similarity heuristic).
- *fastText* can create OOV embeddings on the fly.

Further Reading

- Deep Learning, NLP, and Representations – C. Olah [Olah, 2015]
- Visualizing Representations – C. Olah
- Word Embeddings in 2017: Trends and future directions – S. Ruder
- On Word Embeddings – S. Ruder

Advanced topics

Can we pre-train more?

- In computer vision multiple layers are pre-trained as a model initialization.
- Transfer learning was successfully applied in NLP.

- Using state-of-the-art language models instead of simple word embeddings seems to be a trend: BERT, ELMo, ULMFiT, etc.
- For try-hards only.

What about sub-words?

- Sub-words are a part of fastText.
- We can use them to:
  - model words (or even combine them with word embeddings)
  - model the entire text (Google Translate does this)
- Empirically improve performance, but they are computationally more demanding than word embeddings.

Bibliography

- Collobert & Weston 2008 - A unified architecture for natural language processing: deep neural networks with multitask learning. ICML 2008, Helsinki, Finland.
- Bengio, Ducharme, Vincent & Janvin 2003 – A neural probabilistic language model. JMLR. 3 – 2003.
- Mikolov, Sutskever, Chen, Corrado & Dean 2013a - Distributed Representations of Words and Phrases and their Compositionality. NIPS 2013, Nevada, USA.
- Mikolov, Chen, Corrado & Dean 2013b - Efficient Estimation of Word Representations in Vector Space. ArXiv 2013.
- Olah, 2015 - Deep Learning, NLP, and Representations. Personal blog 2015.
- Mikolov, Yih & Zweig 2013c - Linguistic regularities in continuous space word representations. NAACL 2013, Atlanta, USA.
- Hamilton, Leskovec & Jurafsky 2016 -Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. ACL 2016, Berlin, Germany.
- Garg, Schiebinger, Juragsky & Zou 2017 - Word Embeddings Quantify 100 Years of Gender and Ethnic Stereotypes. ArXiv 2017.
- Iyyer, Manjunatha, Boyd-Graber & Daumé III - Deep unordered composition rivals syntactic methods for text classification. ACL 2015, Beijing, China.
- Pennington, Socher & Manning 2014 - GloVe: Global Vectors for Word Representation. EMNLP 2014, Edinburgh, UK.
- Bojanowski, Grave, Joulin & Mikolov 2017 - Enriching Word Vectors with Subword Information. TACL. 5 – 2017.