

Deep Learning in Applications

Radoslav Neychev
Anastasia Ianina

Harbour.Space University
13.06.2022

Course syllabus:

2 main blocks:

Course syllabus:

2 main blocks:

1. Natural Language Processing
 - a. Language models
 - b. Text generation
 - c. Neural machine translation

Course syllabus:

2 main blocks:

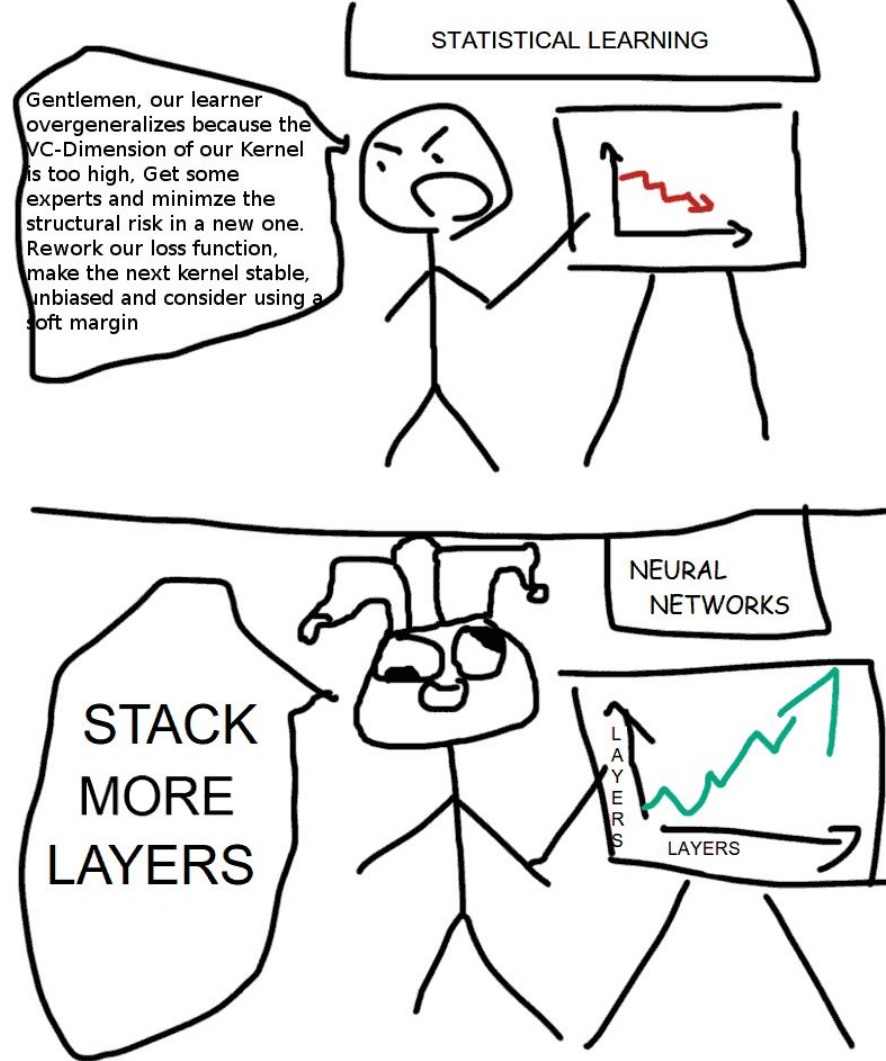
1. Natural Language Processing
2. Reinforcement Learning
 - a. Simple approaches to non-gradient optimization
 - b. Q-learning, SARSA
 - c. DQN
 - d. REINFORCE, AAC

Course syllabus:

2 main blocks:

1. Natural Language Processing
2. Reinforcement Learning

All flavored with Deep Learning



Technical stuff

- Python 3.6+
 - Miniconda is recommended for env managing
- Supported platforms: Linux/macOS/docker
 - Anything else on your own risk
- Course chat in Telegram
- All materials are available at github

This course is using materials and generally based on such courses as:

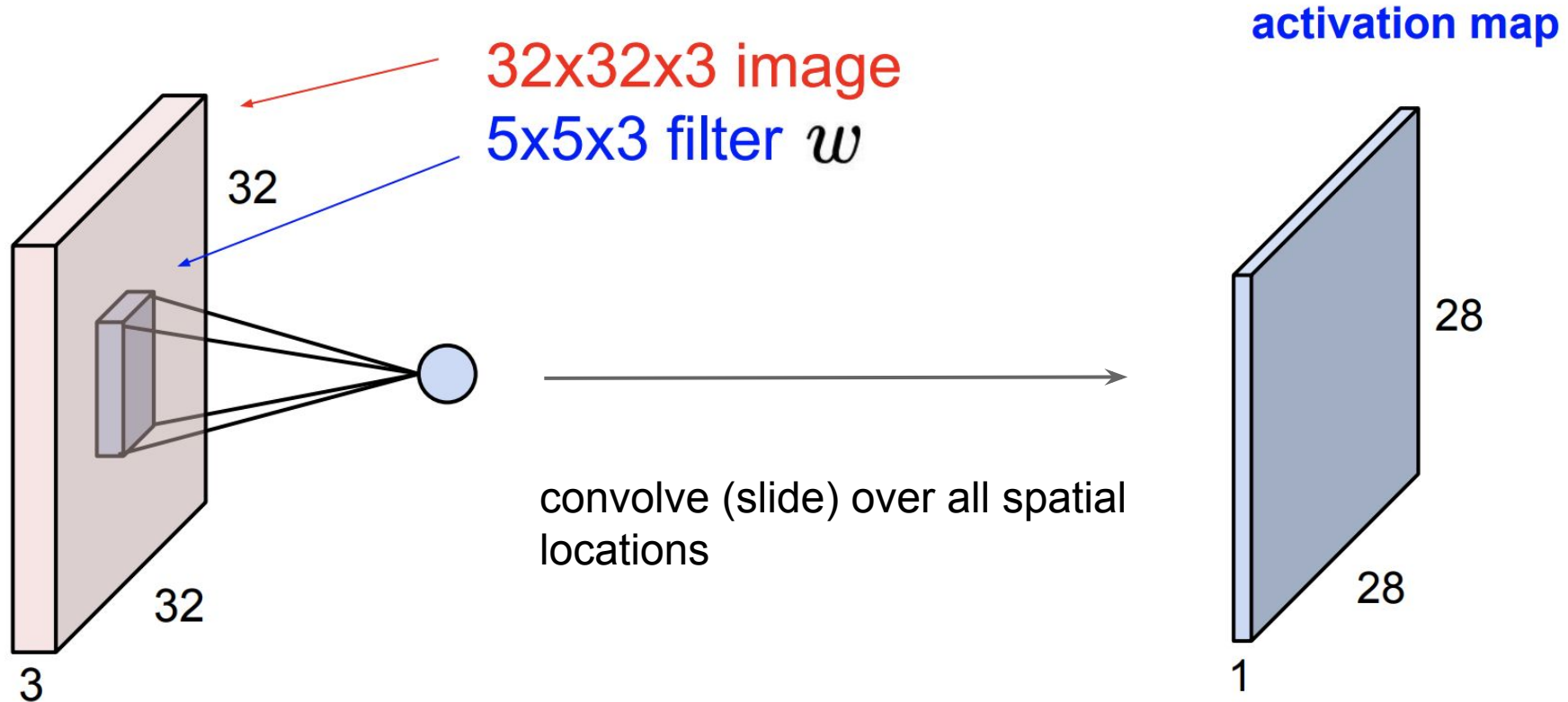
- Stanford:
 - [CS224n](#) Natural Language Processing
 - [CS234n](#) Reinforcement Learning
- Yandex School of Data Analysis:
 - [Practical RL](#)
 - [NLP course](#)
- Berkeley:
 - [CS188x](#) Intro to AI
 - [CS294-112](#) Deep Reinforcement Learning

Special thanks to the teams for developing the materials and making them available online



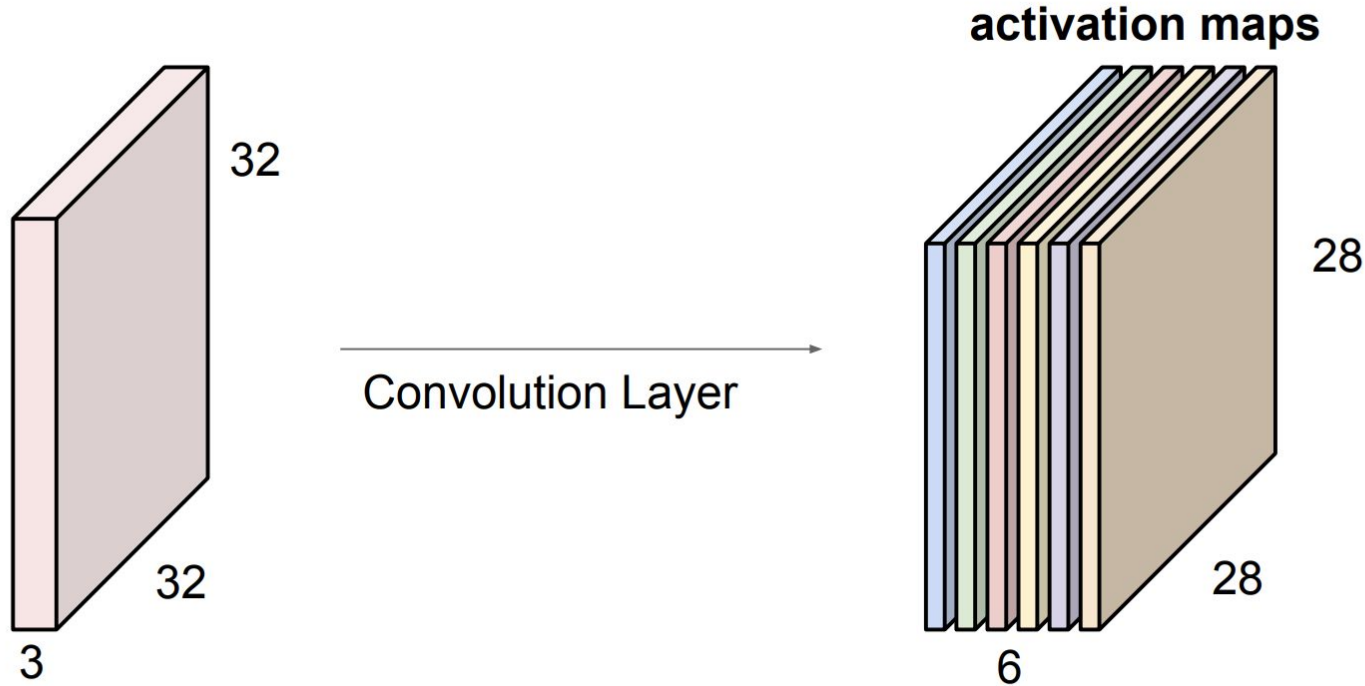
Recap so far

Convolutional layer



Convolutional layer

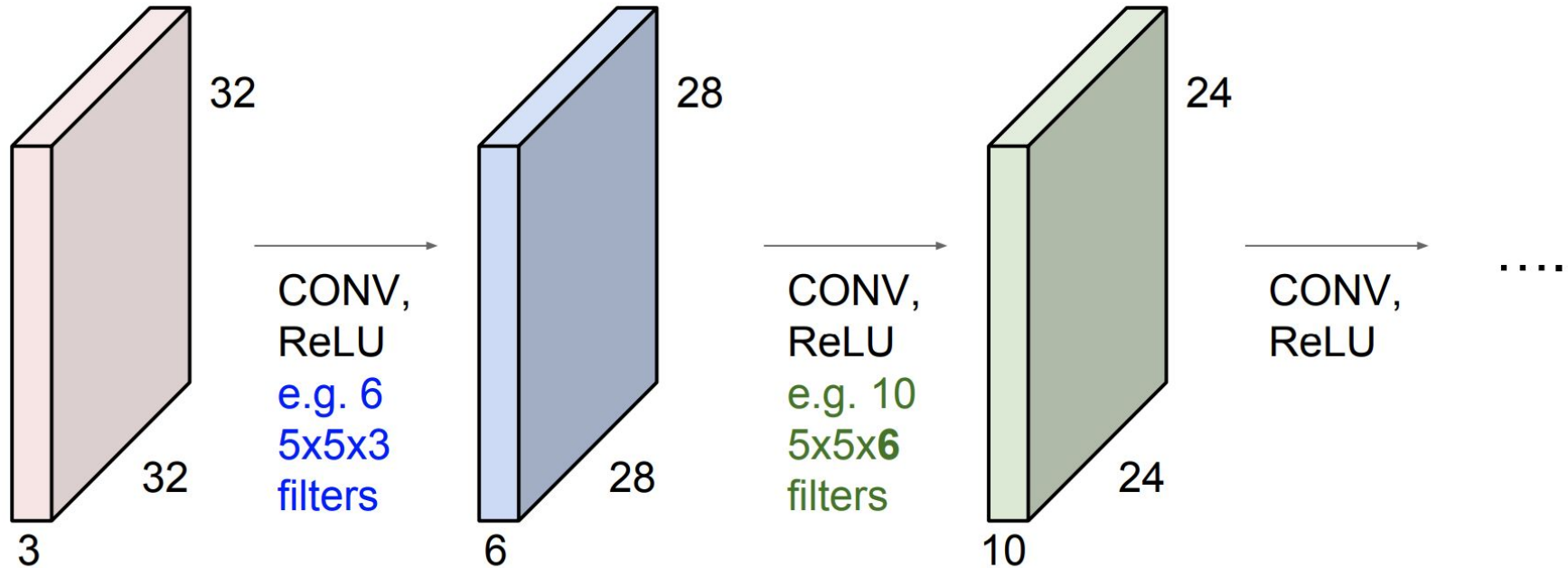
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a “new image” of size 28x28x6!

Convolutional layer

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



RNNs generating...

Shakespeare

PANDARUS:
Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.

Algebraic Geometry (Latex)

Proof. Omitted. □

Lemma 0.1. *Let \mathcal{C} be a set of the construction.*
Let \mathcal{C} be a gerber covering. Let \mathcal{F} be a quasi-coherent sheaves of \mathcal{O} -modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{C})$$

Proof. This is an algebraic space with the composition of sheaves \mathcal{F} on $X_{\text{étale}}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{ \text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F}) \}$$

where \mathcal{G} defines an isomorphism $\mathcal{F} \rightarrow \mathcal{F}$ of \mathcal{O} -modules. □

Lemma 0.2. *This is an integer \mathbb{Z} is injective.* □

Proof. See Spaces, Lemma ??.

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset X$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*
The following to the construction of the lemma follows.
Let X be a scheme. Let X be a scheme covering. Let

$$b: X \rightarrow Y' \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$

be a morphism of algebraic spaces over S and Y .

Proof. Let X be a nonzero scheme of X . Let X be an algebraic space. Let \mathcal{F} be a quasi-coherent sheaf of \mathcal{O}_X -modules. The following are equivalent

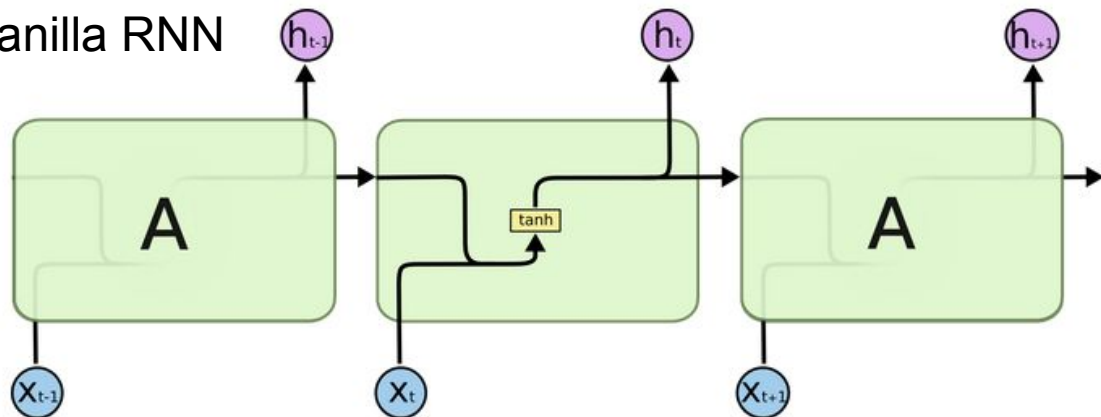
- (1) \mathcal{F} is an algebraic space over S .
- (2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

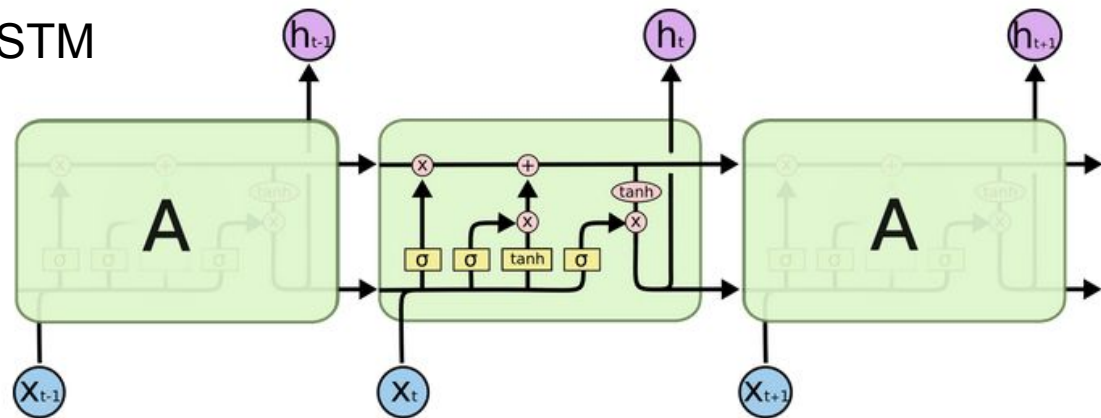
Linux kernel (source code)

```
/*  
 * If this error is set, we will need anything right after that BSD.  
 */  
  
static void action_new_function(struct s_stat_info *wb)  
{  
    unsigned long flags;  
    int lel_idx_bit = e->add, *sys & -((unsigned long) *FIRST_COMPAT);  
    buf[0] = 0xffffffff & (bit << 4);  
    min(inc, slist->bytes);  
    printk(KERN_WARNING "Memory allocated %02x/%02x, "  
           "original MLL instead\n"),  
           min(min(multi_run - s->len, max) * num_data_in),  
           frame_pos, sz + first_seg);  
    div_u64_w(val, inb_p);  
    spin_unlock(&disk->queue_lock);  
    mutex_unlock(&s->sock->mutex);  
    mutex_unlock(&func->mutex);  
    return disassemble(info->pending_bh);  
}
```

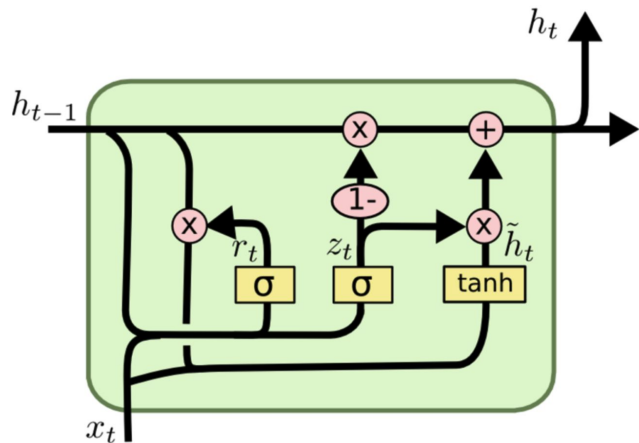
Vanilla RNN



LSTM



GRU



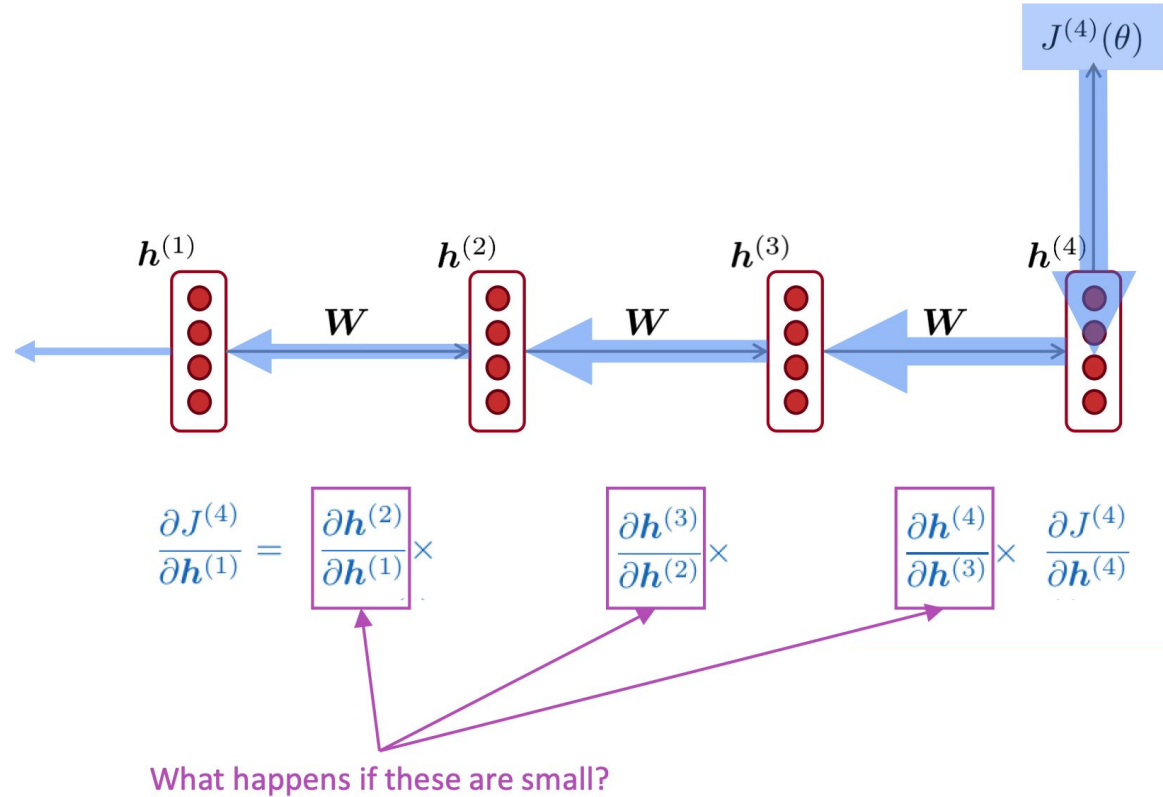
- LSTM and GRU are both great
 - GRU is quicker to compute and has fewer parameters than LSTM
 - There is no conclusive evidence that one consistently performs better than the other
 - LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data)

Rule of thumb: start with LSTM, but switch to GRU if you want something more efficient

Vanishing gradient

Vanishing gradient problem:

When the derivatives are small, the gradient signal gets smaller and smaller as it backpropagates further



More info: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013

<http://proceedings.mlr.press/v28/pascanu13.pdf>

Vanishing gradient in non-RNN

Vanishing gradient is present in **all** deep neural network architectures.

- Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small during backpropagation
- Lower levels are hard to train and are trained slower
- **Potential solution:** or skip-connections/dense-connections/other shortcuts

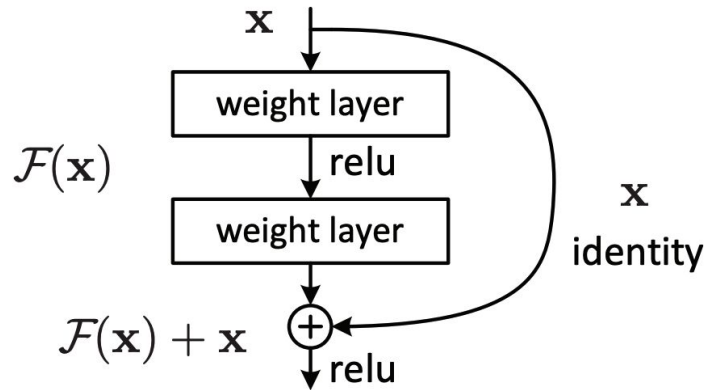
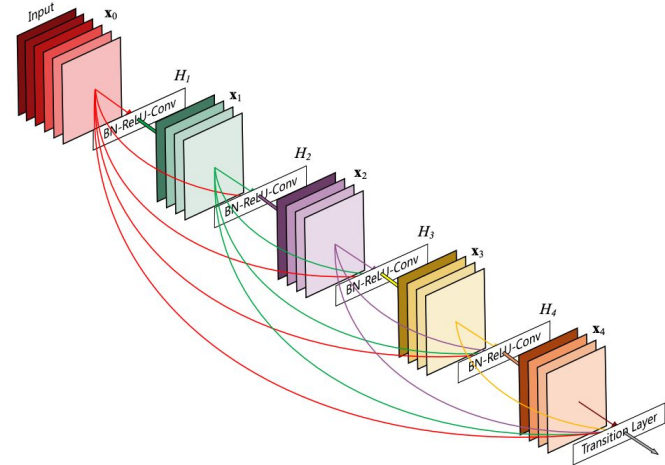


Figure 2. Residual learning: a building block.



Exploding gradient problem

- If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_{\theta} J(\theta)}_{\text{gradient}}$$

- This can cause bad updates: we take too large a step and reach a bad parameter configuration (with large loss)
- In the worst case, this will result in Inf or NaN in your network (then you have to restart training from an earlier checkpoint)

Exploding gradient solution

- Gradient clipping: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

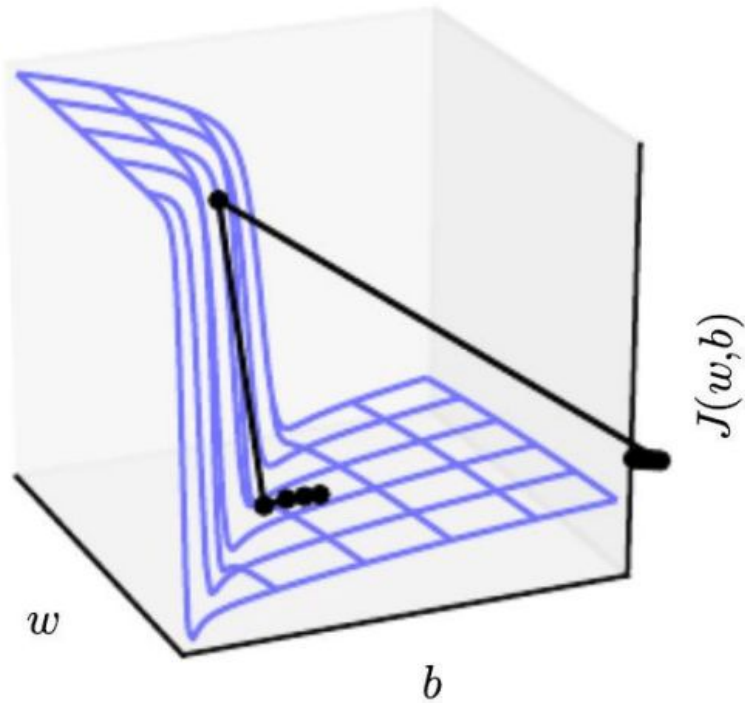
Algorithm 1 Pseudo-code for norm clipping

```
 $\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$   
if  $\|\hat{\mathbf{g}}\| \geq threshold$  then  
     $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$   
end if
```

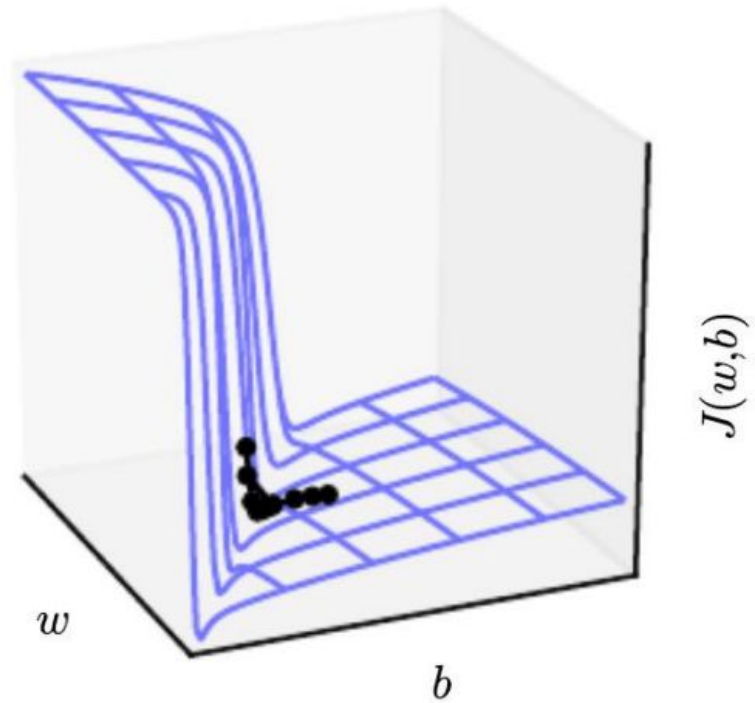
- Intuition: take a step in the same direction, but a smaller step

Exploding gradient solution

Without clipping



With clipping



$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$$

Adding some extra term to the loss function.

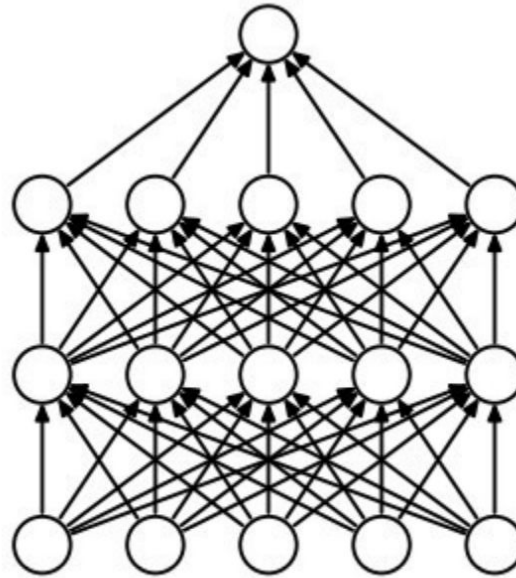
Common cases:

- L2 regularization: $R(W) = \|W\|_2^2$
- L1 regularization: $R(W) = \|W\|_1$
- Elastic Net (L1 + L2): $R(W) = \beta \|W\|_2^2 + \|W\|_1$

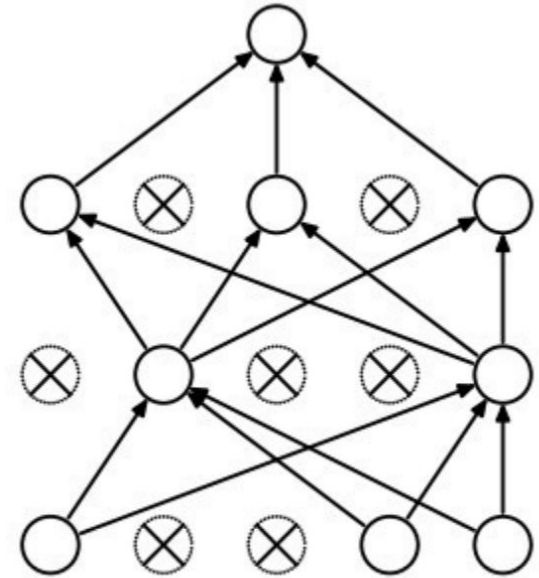
Regularization: Dropout

Some neurons are “dropped” during training.

Prevents overfitting.



(a) Standard Neural Net

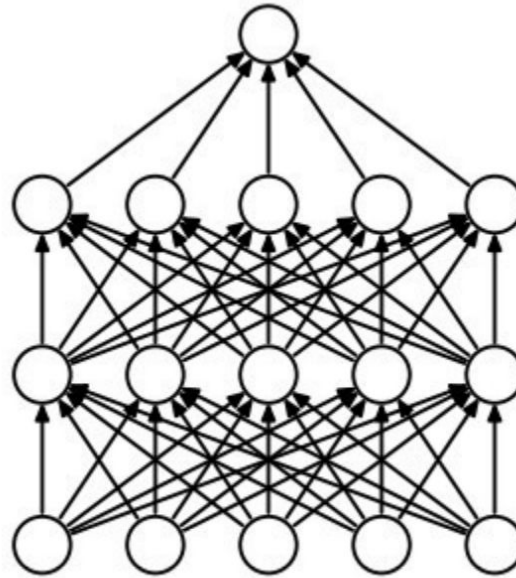


(b) After applying dropout.

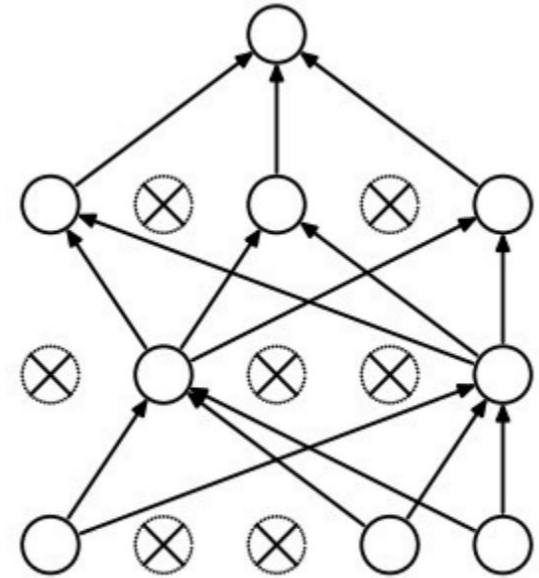
Regularization: Dropout

Some neurons are “dropped” during training.

Prevents overfitting.



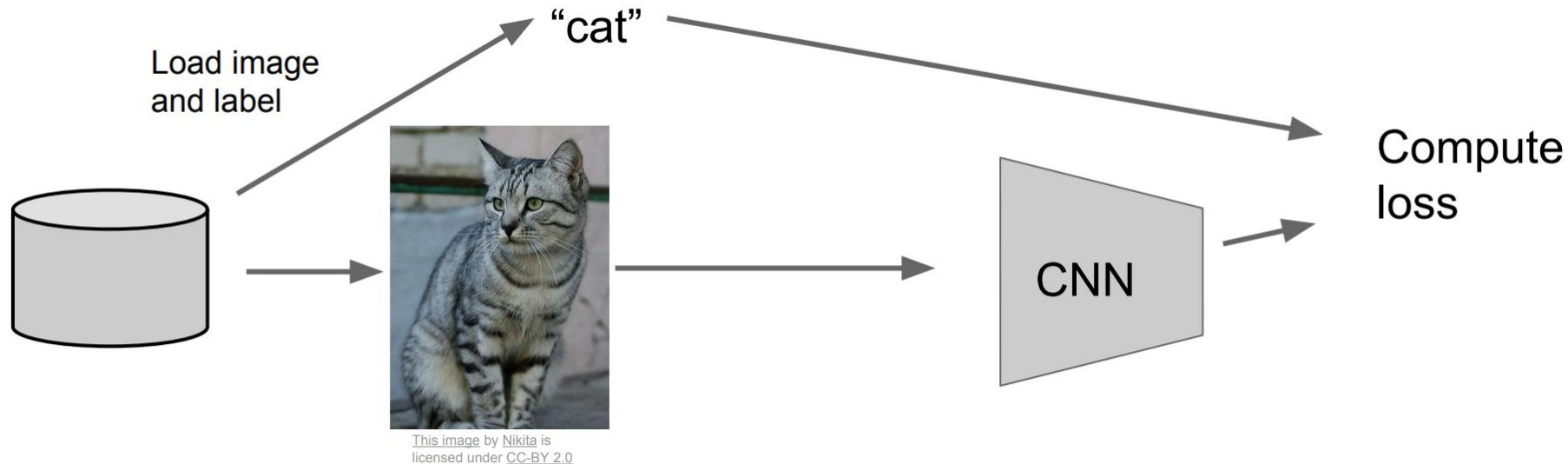
(a) Standard Neural Net



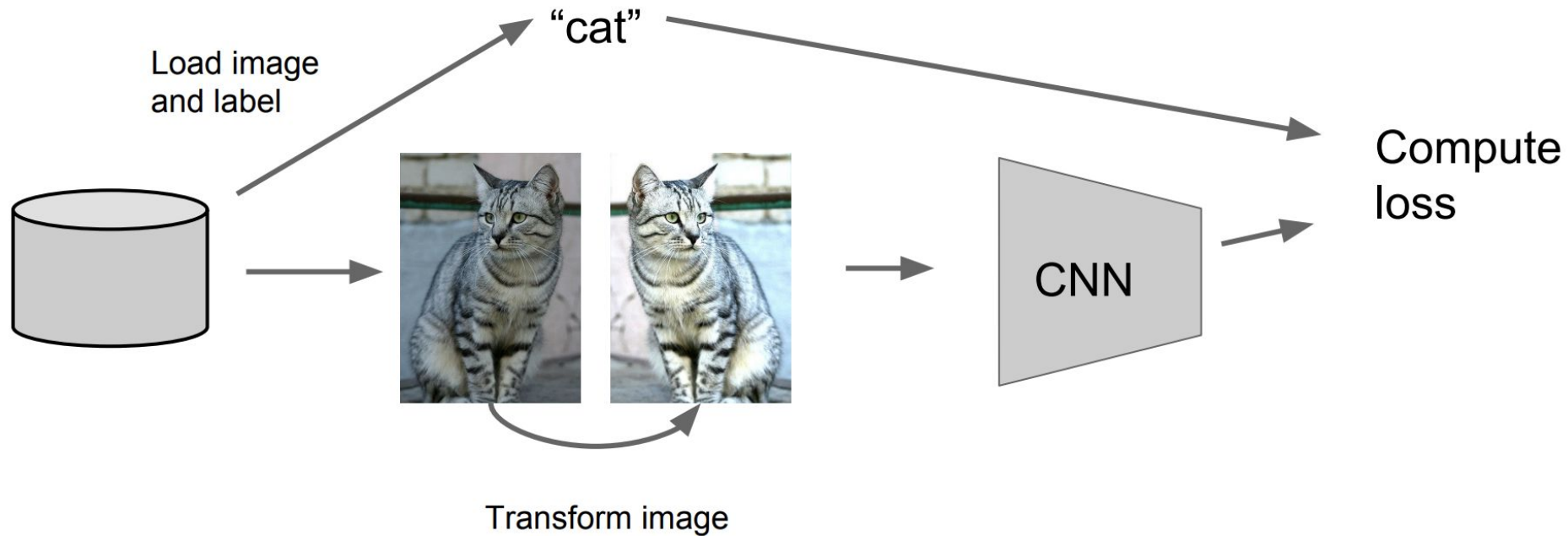
(b) After applying dropout.

Actually, on test case output should be normalized. See sources for more info.

Regularization: data augmentation



Regularization: data augmentation



Optimization:

- Adam is great to start
 - Initial learning rate $3e-4$
- Momentum is great
- Remember the learning rate decay

Regularization:

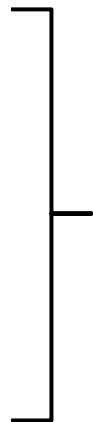
- Add some weight constraints
- Add some random noise during train and marginalize it during test
- Add some prior information in appropriate form

Natural Language Processing: Introduction

- NLP: introduction
- Text Preprocessing
- Feature Extraction: classical approach
 - Bag-of-Words
 - Bag-of-Ngramms
 - TF-IDF
- Word Embeddings

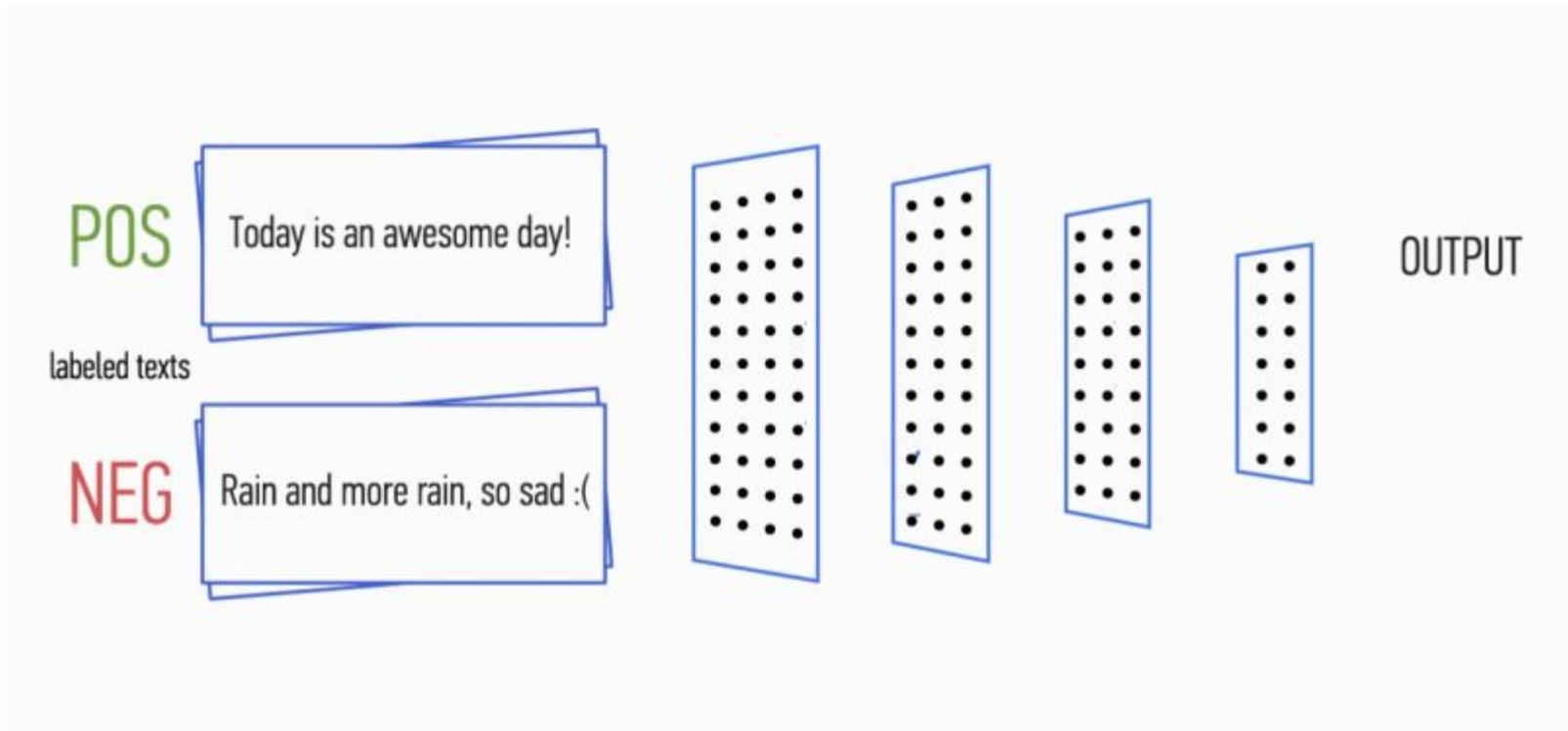
Popular NLP tasks

- Sentiment analysis
- Spam filtering
- Fake news detection
- Topic prediction
- #hashtag prediction



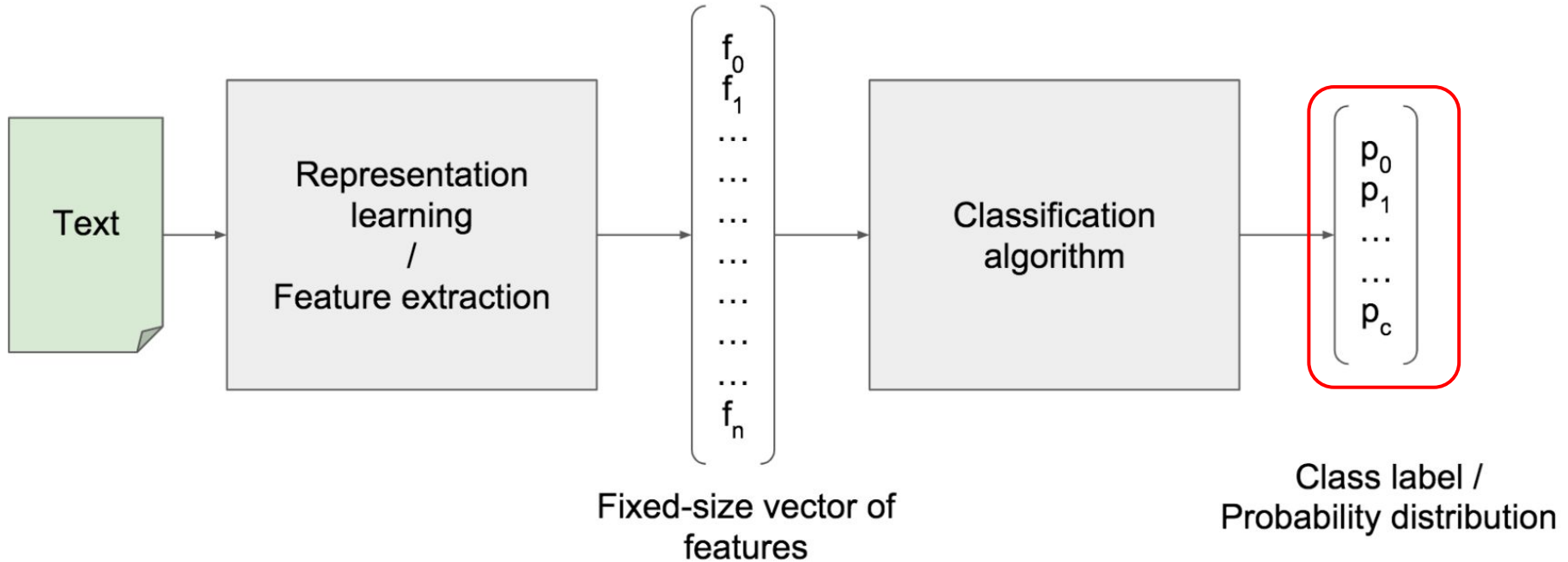
Text classification tasks

Example: sentiment analysis

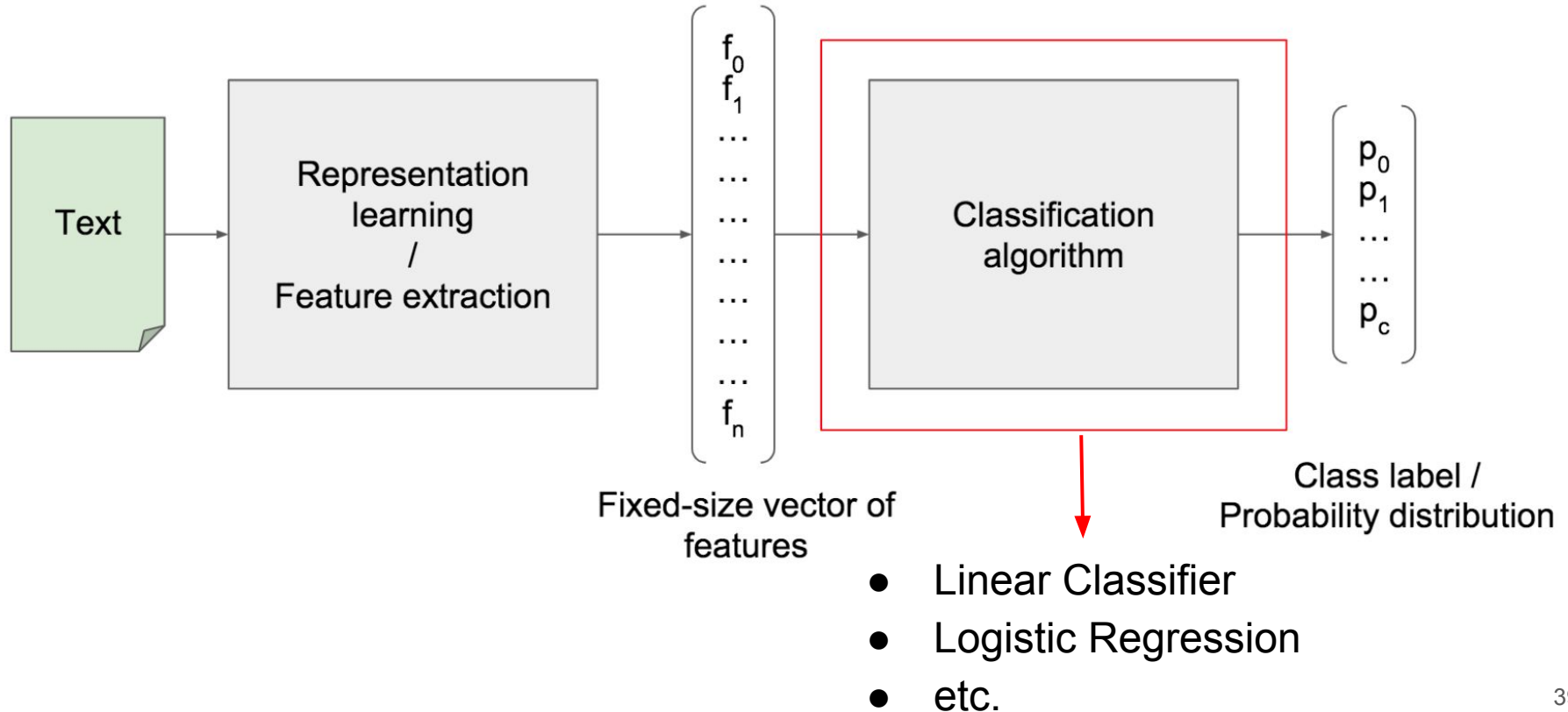


- Discrete labels:
 - Binary
 - spam filtering, sentiment analysis
 - Multi-class
 - categorization of items by its description
 - Multi-label
 - #hashtag prediction
- Continuous labels:
 - Predict product price by its description

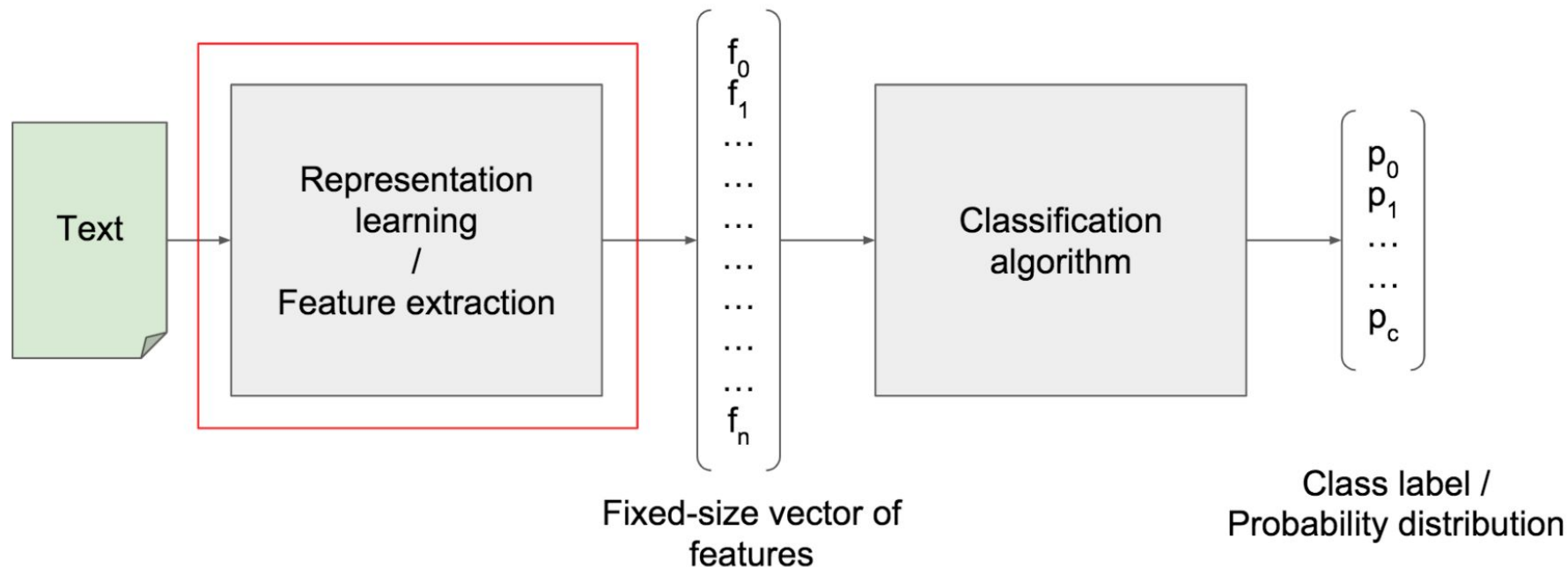
Text classification in general



Text classification in general

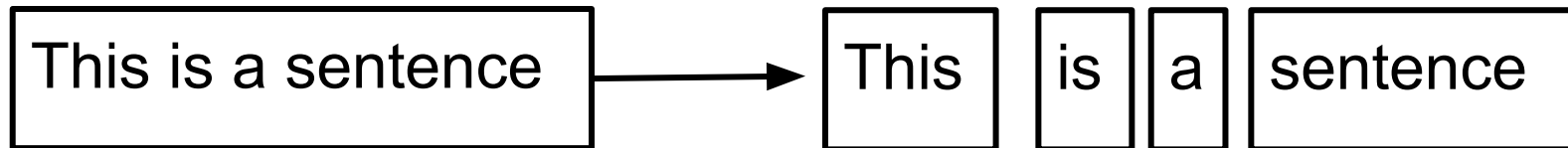


Text classification in general



Feature extraction

- Tokenization: split the input into tokens



the dog is on the table

0	0	1	1	0	1	1	1
are	cat	dog	is	now	on	table	the

- Problems:
 - No information about words order
 - Word vectors are huge and very sparse
 - Word vectors are not normalized
 - Same words can take different forms

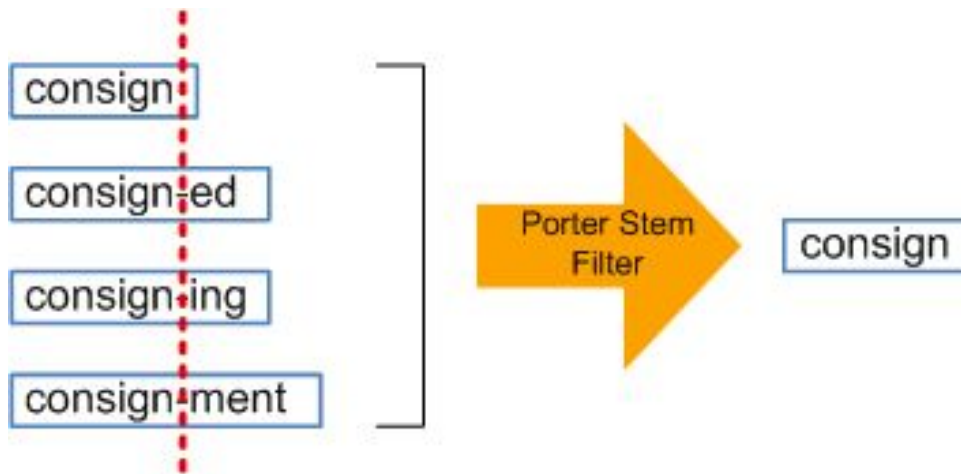
Text Preprocessing

- Token normalization

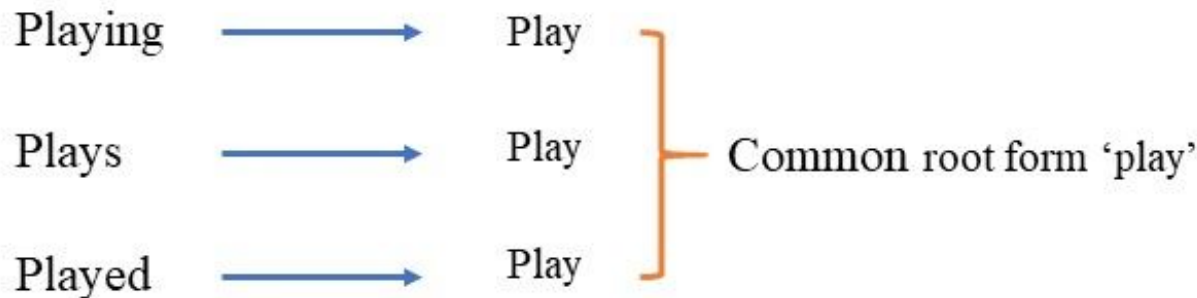
Dog, dogs → dog

Bark, barks → bark

- Token normalization:
 - **Stemming**: removing and replacing suffixes to get to the root of the word (**stem**)



- Token normalization:
 - **Stemming**: removing and replacing suffixes to get to the root of the word (**stem**)
 - **Lemmatization**: to get base or dictionary form of a word (**lemma**)



Stemming: Porter vs Lancaster

Porter stemmer

- Published in 1979
- Base starting option

Snowball stemmer (Porter 2)

- Based on Porter
- More aggressive
- Most popular option now

Lancaster stemmer

- Published in 1990
- The most aggressive
- Easy adding of your own rules

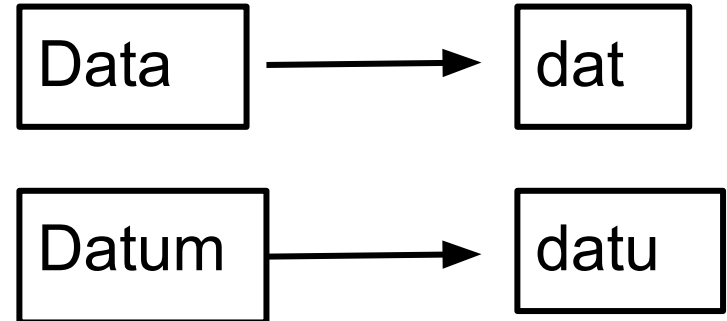
- Porter's stemmer:
 - **Heuristics, applied one-by-one:**
 - SSES - SS (dresses - dress)
 - IES - I (ponies - poni)
 - S - <empty> (dogs - dog)
 - **What's wrong?**
 - **Overstemming and understemming**

Overstemming

- University
- Universal
- Universities
- Universe

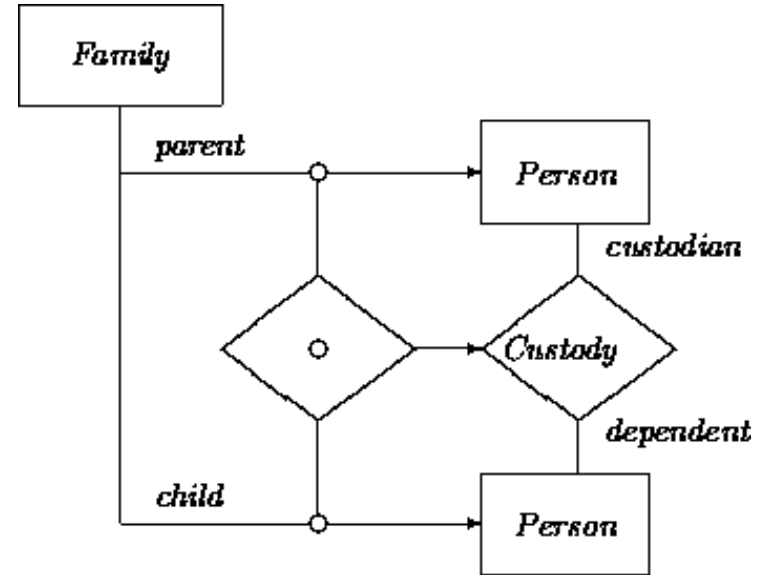
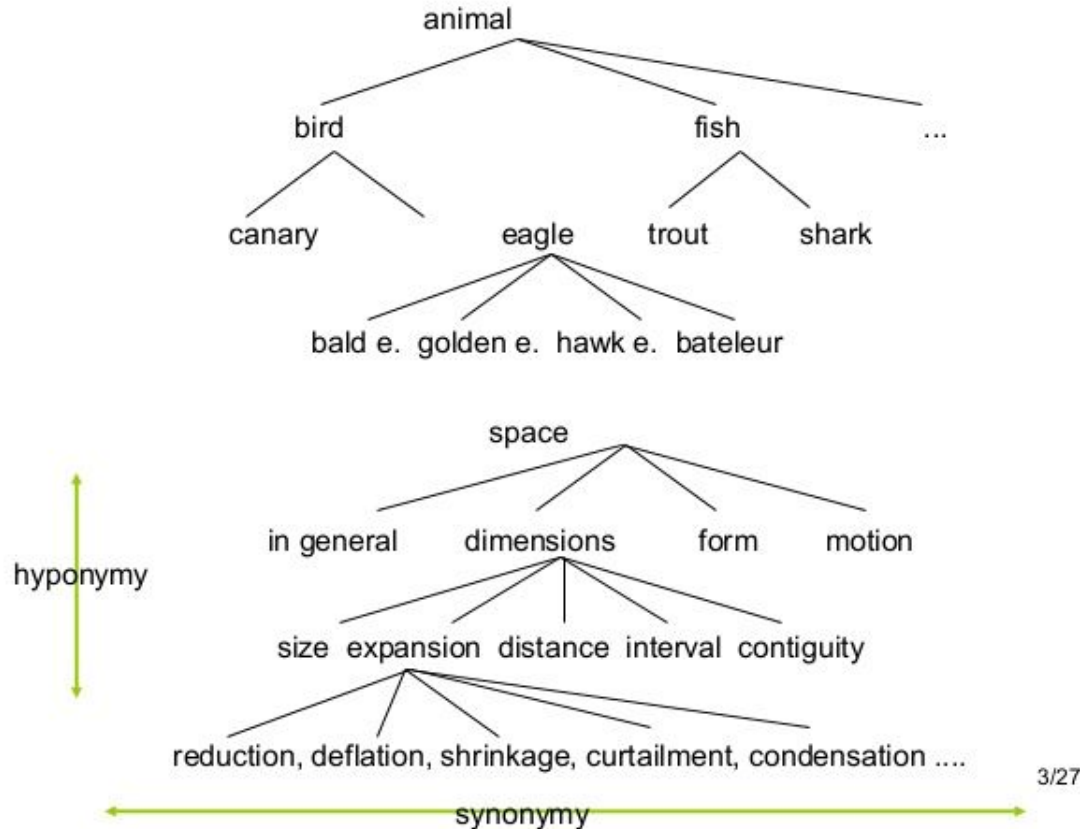
} Univers

Understemming



- Lemmatizer from NLTK:
 - Tries to resolve word to its dictionary form
 - Based on **WordNet** database
 - For the best results feed part-of-speech tagger

BTW, what is WordNet?



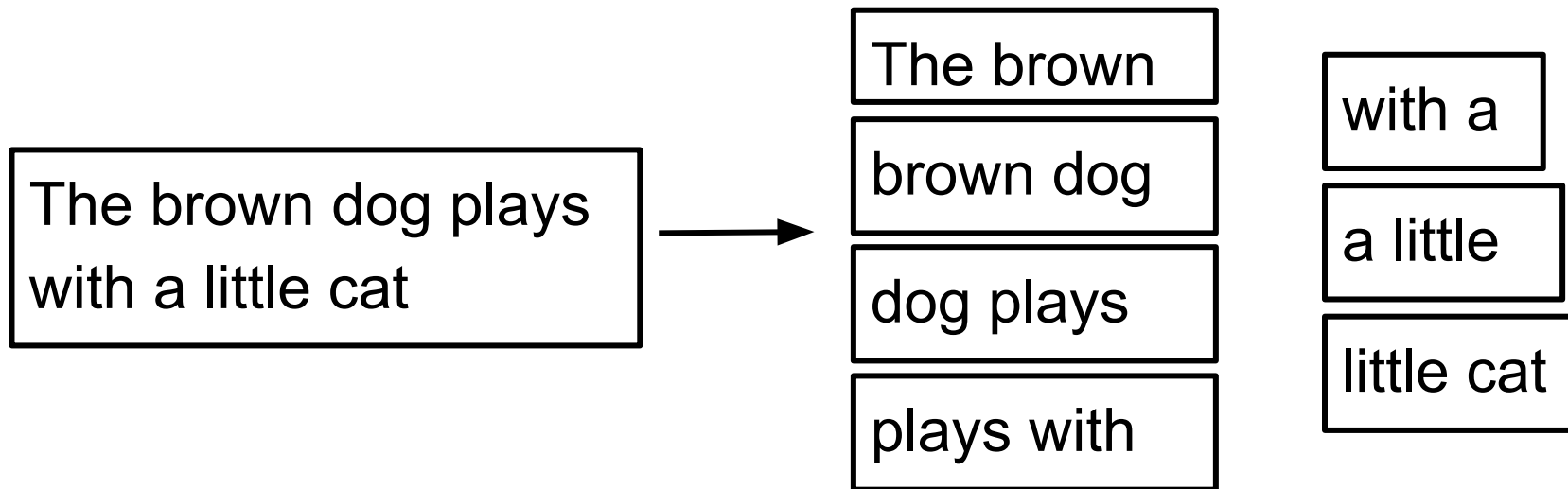
Handful tools for preprocessing

- [NLTK](#)
 - `nltk.stem.SnowballStemmer`
 - `nltk.stem.PorterStemmer`
 - `nltk.stem.WordNetLemmatizer`
 - `nltk.corpus.stopwords`
- [BeautifulSoup](#) (for parsing HTML)
- Regular Expressions (`import re`)
- [Pymorphy2](#)

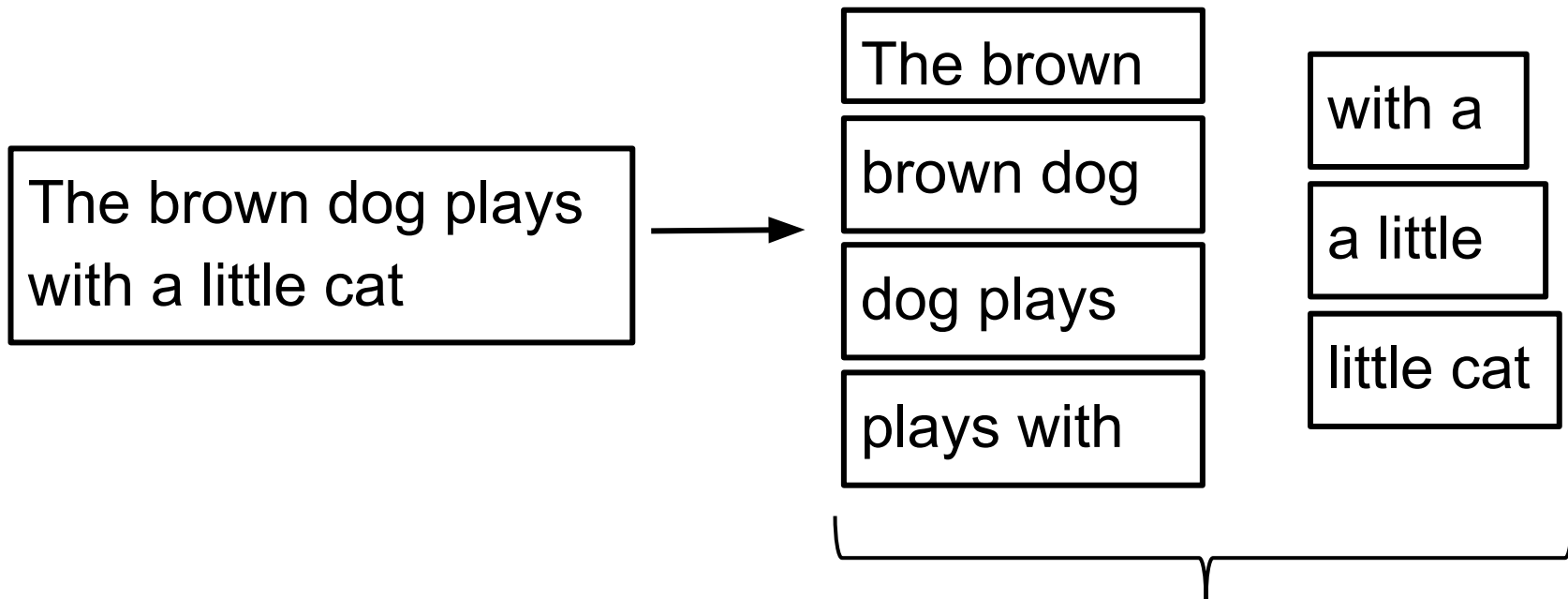
What's left?

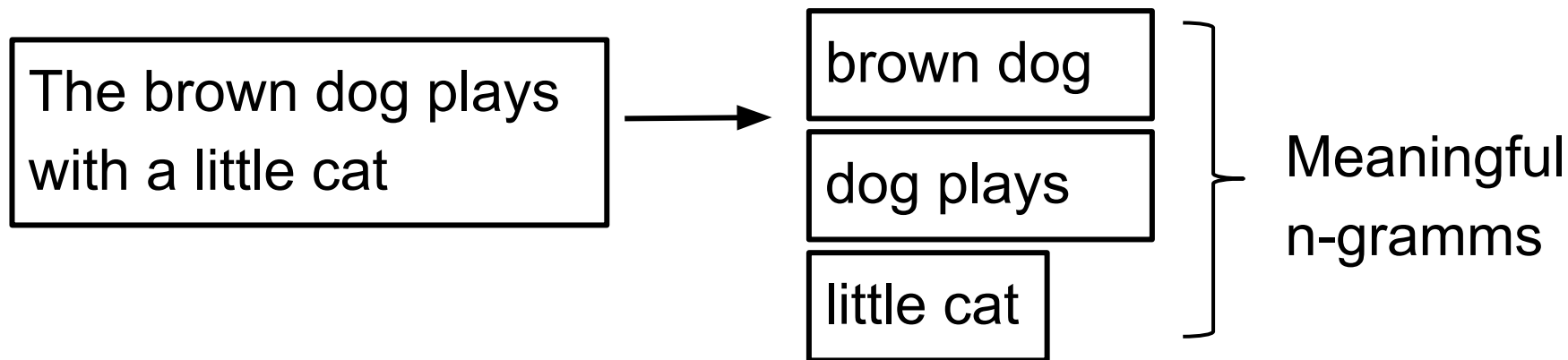
- Capital Letters
- Punctuation
- Contractions (e.g, etc.)
- Numbers (dates, ids, page numbers)
- Stop-words (“the”, “is”, etc.)
- Tags

- How to improve BOW?
 - Use n-gramms instead of words!



Bag-of-Words





Meaningful n-gramms are often called **collocations**

How to detect meaningful n-gramms?

- Delete:
 - High-frequency n-gramms
 - Articles, prepositions
 - Auxiliary verbs (to be, to have, etc.)
 - General vocabulary
 - Low-frequency n-gramms
 - Typos
 - Combinations that occur 1-2 times in a text

- **Term Frequency (tf):** gives us the frequency of the word in each document in the corpus.

$$\text{tf}(t, d) = f_{t, d}$$

- **Inverse Document Frequency (idf):** used to calculate the weight of rare words across all documents in the corpus. The words that occur rarely in the corpus have a high IDF score.

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

N : total number of documents in the corpus $N = |D|$

$|\{d \in D : t \in d\}|$: number of documents where the term t appears

TF-IDF example

- *Sentence A:* The car is driven on the road.
- *Sentence B:* The truck is driven on the highway.

(each sentence is a separate document)

TF-IDF example

Word	TF		IDF	TF * IDF	
	A	B		A	B
The	1/7	1/7			
Car	1/7	0			
Truck	0	1/7			
Is	1/7	1/7			
Driven	1/7	1/7			
On	1/7	1/7			
The	1/7	1/7			
Road	1/7	0			
Highway	0	1/7			

TF-IDF example

Word	TF		IDF	TF * IDF	
	A	B		A	B
The	1/7	1/7	$\log(2/2)=0$		
Car	1/7	0	$\log(2/1)=0.3$		
Truck	0	1/7	$\log(2/1)=0.3$		
Is	1/7	1/7	$\log(2/2)=0$		
Driven	1/7	1/7	$\log(2/2)=0$		
On	1/7	1/7	$\log(2/2)=0$		
The	1/7	1/7	$\log(2/2)=0$		
Road	1/7	0	$\log(2/1)=0.3$		
Highway	0	1/7	$\log(2/1)=0.3$		

TF-IDF example

Word	TF		IDF	TF * IDF	
	A	B		A	B
The	1/7	1/7	$\log(2/2)=0$	0	0
Car	1/7	0	$\log(2/1)=0.3$	0.043	0
Truck	0	1/7	$\log(2/1)=0.3$	0	0.043
Is	1/7	1/7	$\log(2/2)=0$	0	0
Driven	1/7	1/7	$\log(2/2)=0$	0	0
On	1/7	1/7	$\log(2/2)=0$	0	0
The	1/7	1/7	$\log(2/2)=0$	0	0
Road	1/7	0	$\log(2/1)=0.3$	0.043	0
Highway	0	1/7	$\log(2/1)=0.3$	0	0.043

TF-IDF example

```
from sklearn.feature_extraction.text  
import TfidfVectorizer
```



Word Embeddings

- **One-hot vectors:**

Rome = $[1, 0, 0, 0, 0, 0, \dots, 0]$

Paris = $[0, 1, 0, 0, 0, 0, \dots, 0]$

Italy = $[0, 0, 1, 0, 0, 0, \dots, 0]$

France = $[0, 0, 0, 1, 0, 0, \dots, 0]$

word V

Problems:

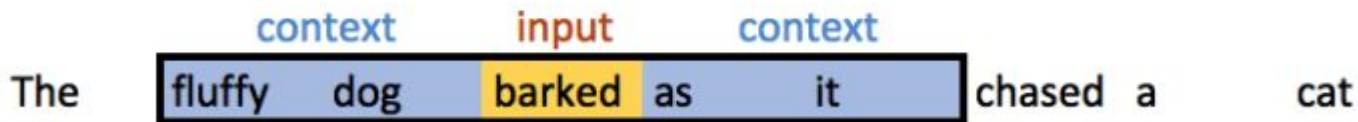
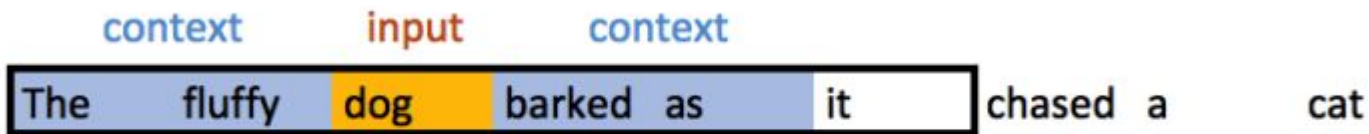
- Huge vectors
- VERY sparse
- No semantics or word similarity information included

Distributional semantics

Does vector similarity imply semantic similarity?

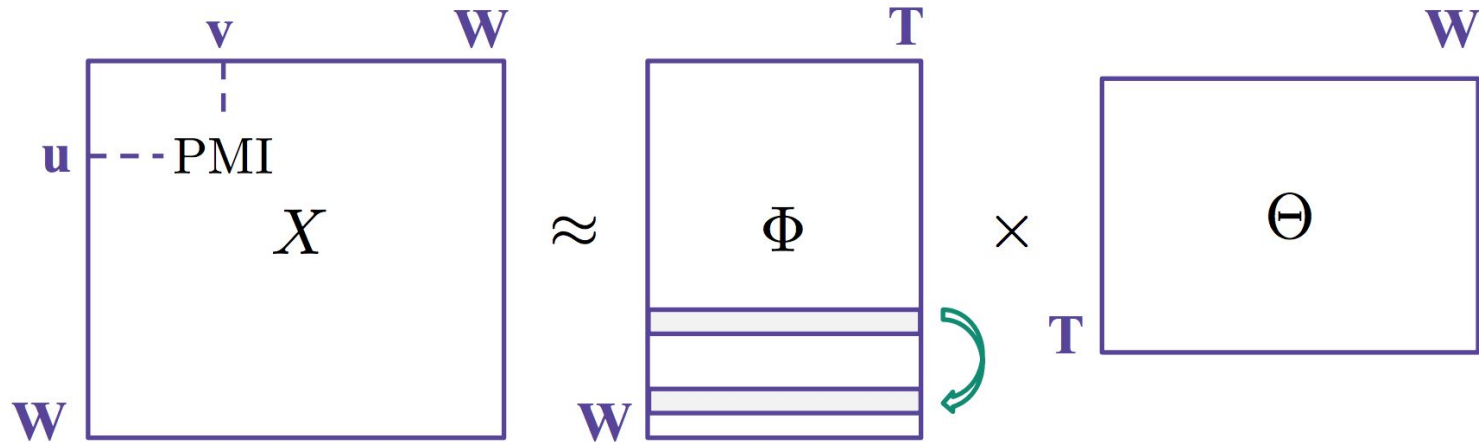
“You shall know a word by the company it keeps”

Firth, 1957



Word representations via matrix factorization

- **Input: PMI, word cooccurrences, etc.**
- **Method: dimensionality reduction (SVD)**
- **Output: word similarities**



- Delete:
 - High-frequency n-gramms
 - Articles, prepositions
 - Auxiliary verbs (to be, to have, etc.)
 - General vocabulary
 - Low-frequency n-gramms
 - Typos
 - Combinations that occur 1-2 times in a text

Collocations: context is all you need

- Cooccurrence counters in a window of fixed size
 - n_{uv} states for the number of times we've seen word u and word v together in the window
- Better solution: Pointwise Mutual Information (PMI)

$$PMI = \log \frac{p(u, v)}{p(u)p(v)} = \log \frac{n_{uv}n}{n_u n_v}$$

- Much better solution: **Positive PMI (pPMI)**

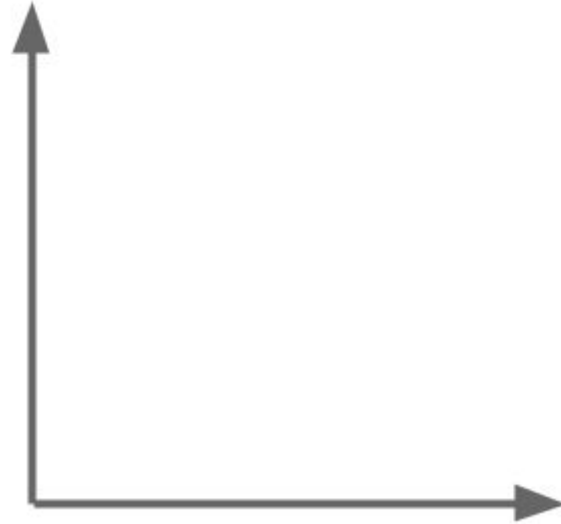
$$pPMI = \max(0, PMI)$$

Frequency With Filter	PMI	T-test With Filter	Chi-Sq Test
(front, desk)	(universal, studios)	(front, desk)	(wi, fi)
(great, location)	(howard, johnson)	(great, location)	(cracker, barrel)
(friendly, staff)	(cracker, barrel)	(friendly, staff)	(howard, johnson)
(hot, tub)	(santa, barbara)	(hot, tub)	(la, quinta)
(clean, room)	(sub, par)	(continental, breakfast)	(front, desk)
(hotel, staff)	(santana, row)	(free, breakfast)	(universal, studios)
(continental, breakfast)	(e, g)	(great, place)	(santa, barbara)
(nice, hotel)	(elk, springs)	(parking, lot)	(santana, row)
(free, breakfast)	(times, square)	(customer, service)	(, more)
(great, place)	(ear, plug)	(desk, staff)	(flat, screen)
(desk, staff)	(la, quinta)	(walk, distance)	(french, quarter)
(parking, lot)	(fire, pit)	(comfortable, bed)	(elk, springs)
(customer, service)	(san, clemente)	(nice, hotel)	(walking, distance)

Why not to learn word vectors?

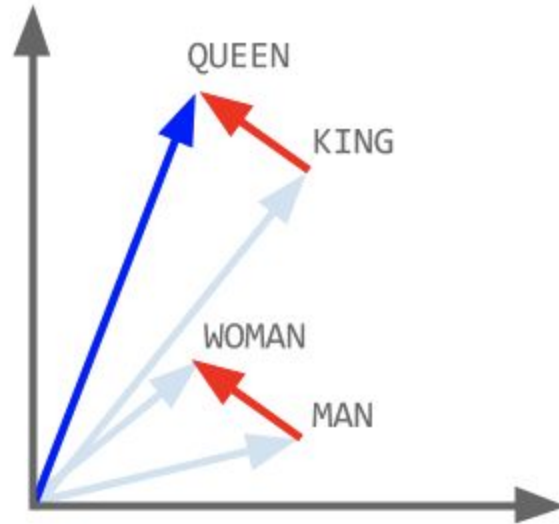
Embeddings: intuition

What is $\text{king} - \text{man} + \text{woman}$?

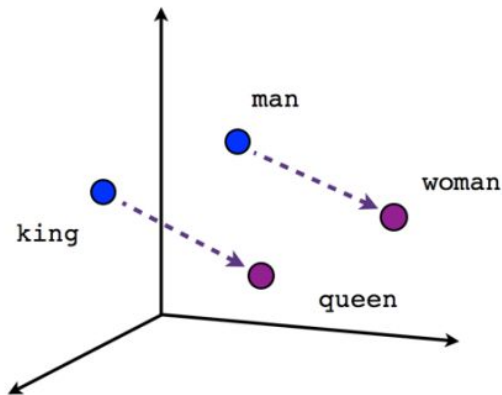


Embeddings: intuition

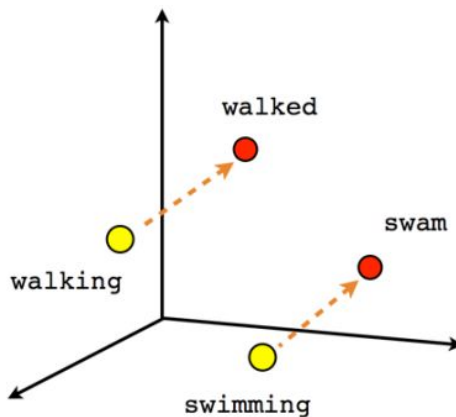
So $\text{king} - \text{man} + \text{woman} = \text{queen!}$



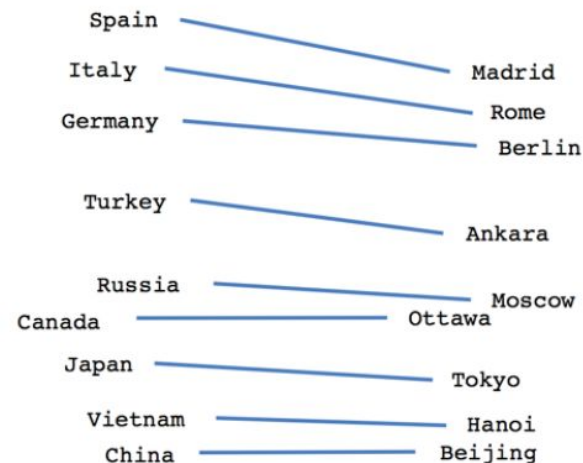
- **Word2vec** (Mikolov et al. 2013) - a framework for learning word embeddings



Male-Female



Verb tense

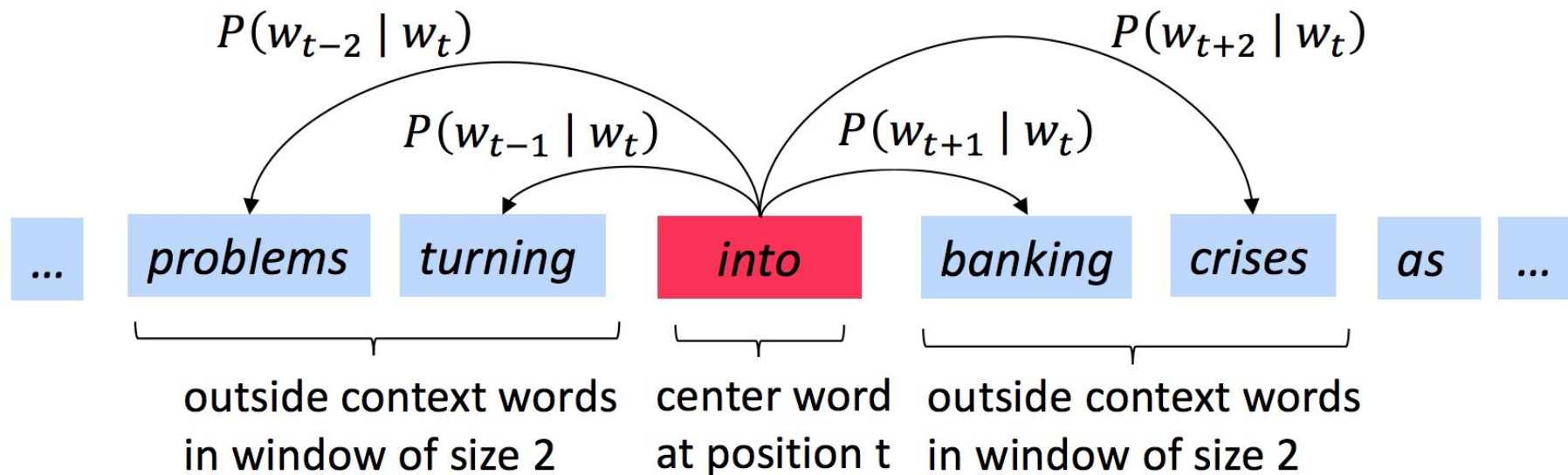


Country-Capital

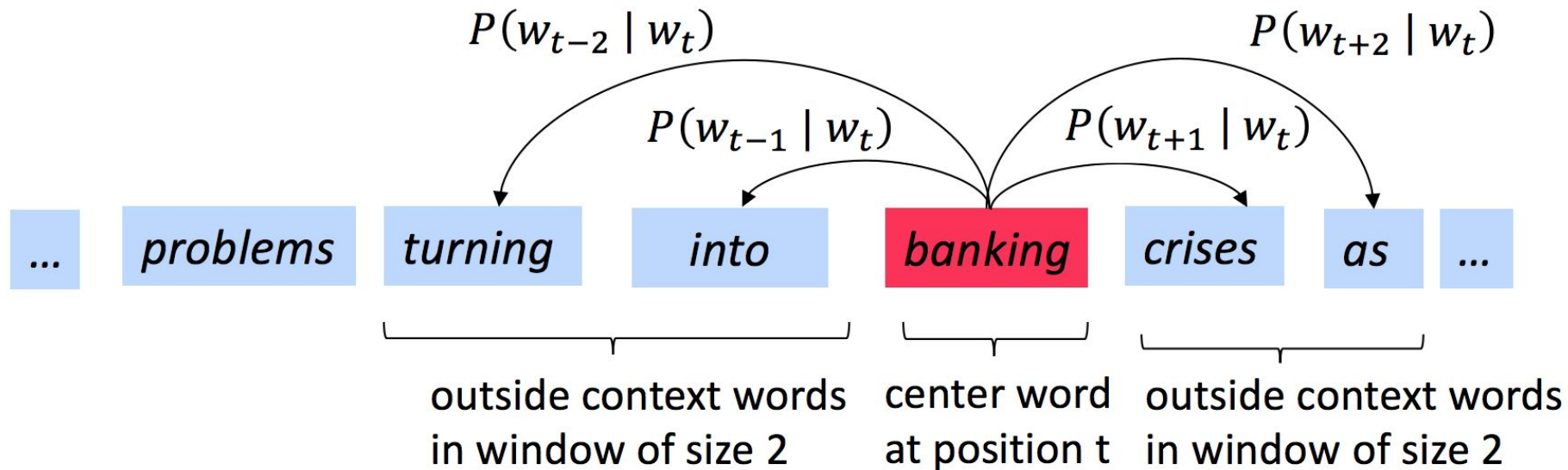
Embeddings: word2vec

Source Text	Training Samples					
<table><tr><td>The</td><td>quick</td><td>brown</td></tr></table> fox jumps over the lazy dog. ➡	The	quick	brown	(the, quick) (the, brown)		
The	quick	brown				
The <table><tr><td>quick</td><td>brown</td><td>fox</td></tr></table> jumps over the lazy dog. ➡	quick	brown	fox	(quick, the) (quick, brown) (quick, fox)		
quick	brown	fox				
The quick <table><tr><td>brown</td><td>fox</td><td>jumps</td></tr></table> over the lazy dog. ➡	brown	fox	jumps	(brown, the) (brown, quick) (brown, fox) (brown, jumps)		
brown	fox	jumps				
The <table><tr><td>quick</td><td>brown</td><td>fox</td><td>jumps</td><td>over</td></tr></table> the lazy dog. ➡	quick	brown	fox	jumps	over	(fox, quick) (fox, brown) (fox, jumps) (fox, over)
quick	brown	fox	jumps	over		

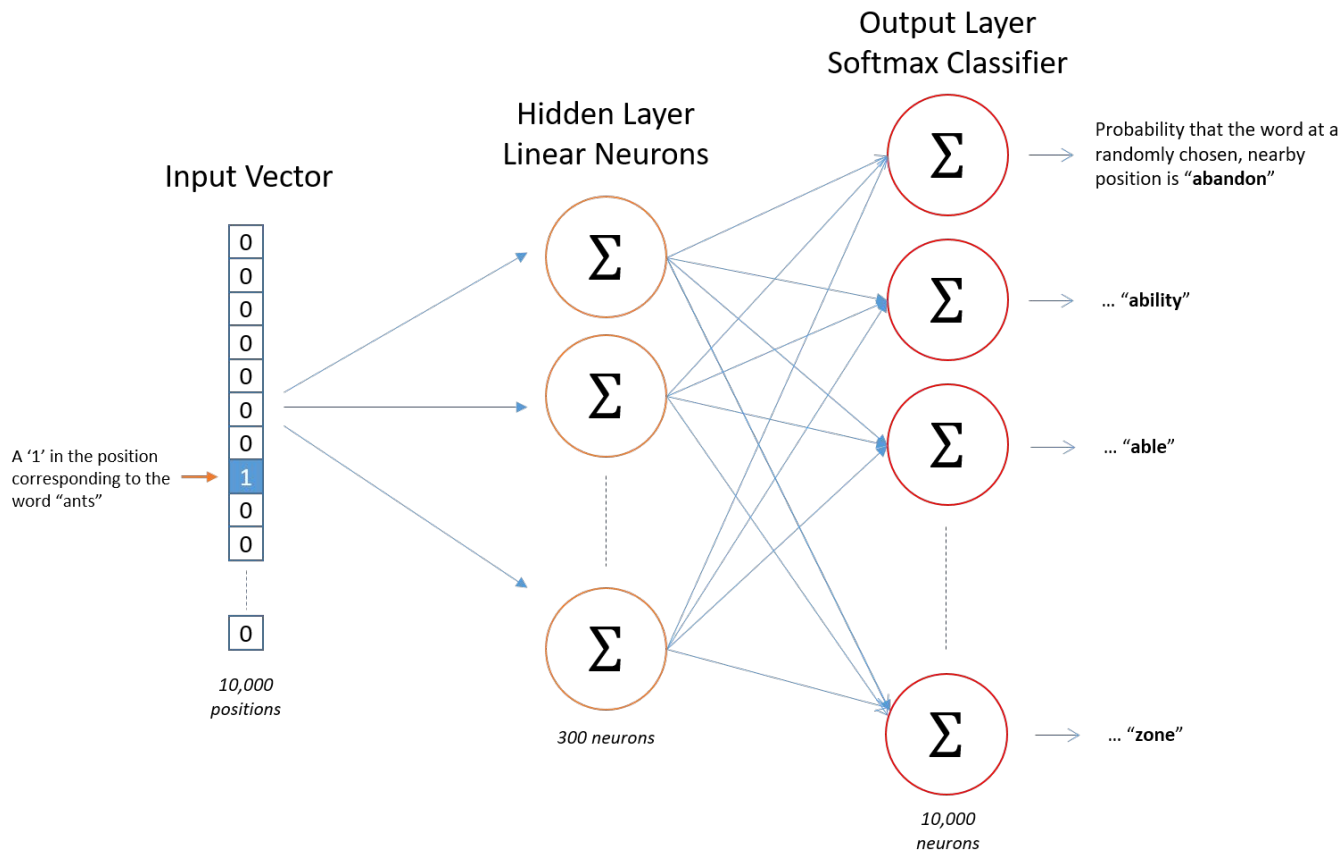
Embeddings: word2vec



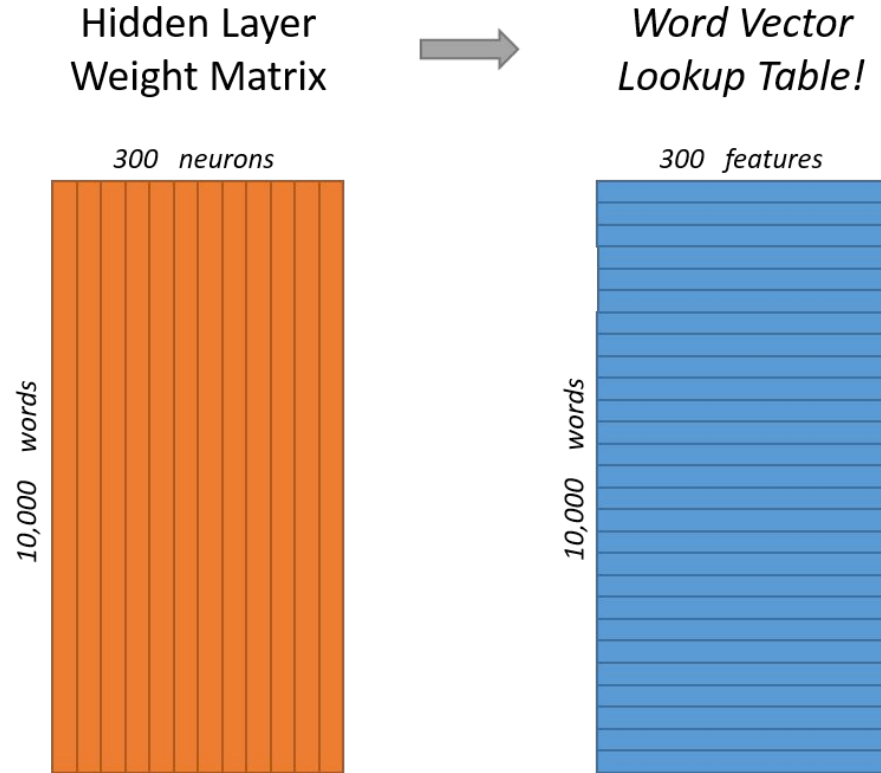
Embeddings: word2vec



Embeddings: word2vec



Embeddings: word2vec



Embeddings: word2vec

- Word vectors with 300 components
- Vocabulary of 10,000 words.
- Weight matrix with $300 \times 10,000 = 3$ million weights each!

Training is too long and computationally expensive

How to fix this?

Basic approaches:

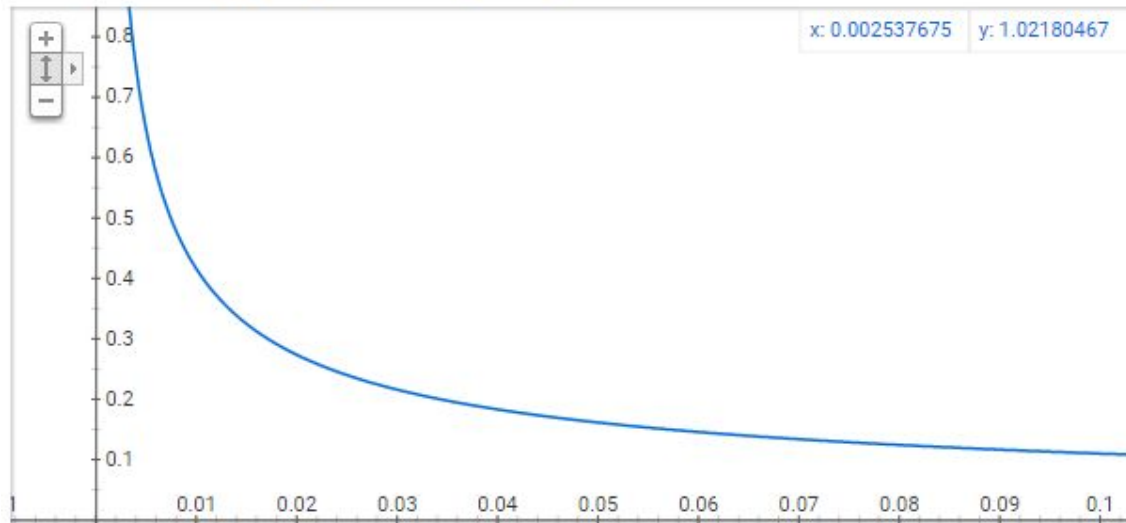
1. Treating common word pairs or phrases as single “words” in their model.
2. Subsampling frequent words to decrease the number of training examples.
3. Modifying the optimization objective with a technique they called “Negative Sampling”, which causes each training sample to update only a small percentage of the model’s weights.

Embeddings: word2vec

Subsampling frequent words.

w_i is the word, $z(w_i)$ is the fraction of this word in the whole text

Graph for $(\sqrt{x/0.001}+1)*0.001/x$



$P(w_i)$ is the probability of *keeping* the word:

$$P(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1 \right) \cdot \frac{0.001}{z(w_i)}$$

Embeddings: negative sampling

Negative Sampling idea: only few words error is computed. All other words have zero error, so no updates by the backprop mechanism.

More frequent words are selected to be negative samples more often. The probability for selecting a word is just its weight divided by the sum of weights for all words.

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^n (f(w_j)^{3/4})}$$

Word2vec: two models

Continuous BOW (CBOW)

$$p(w_i | w_{i-h}, \dots, w_{i+h})$$

Predict center word from
(bag of) context words

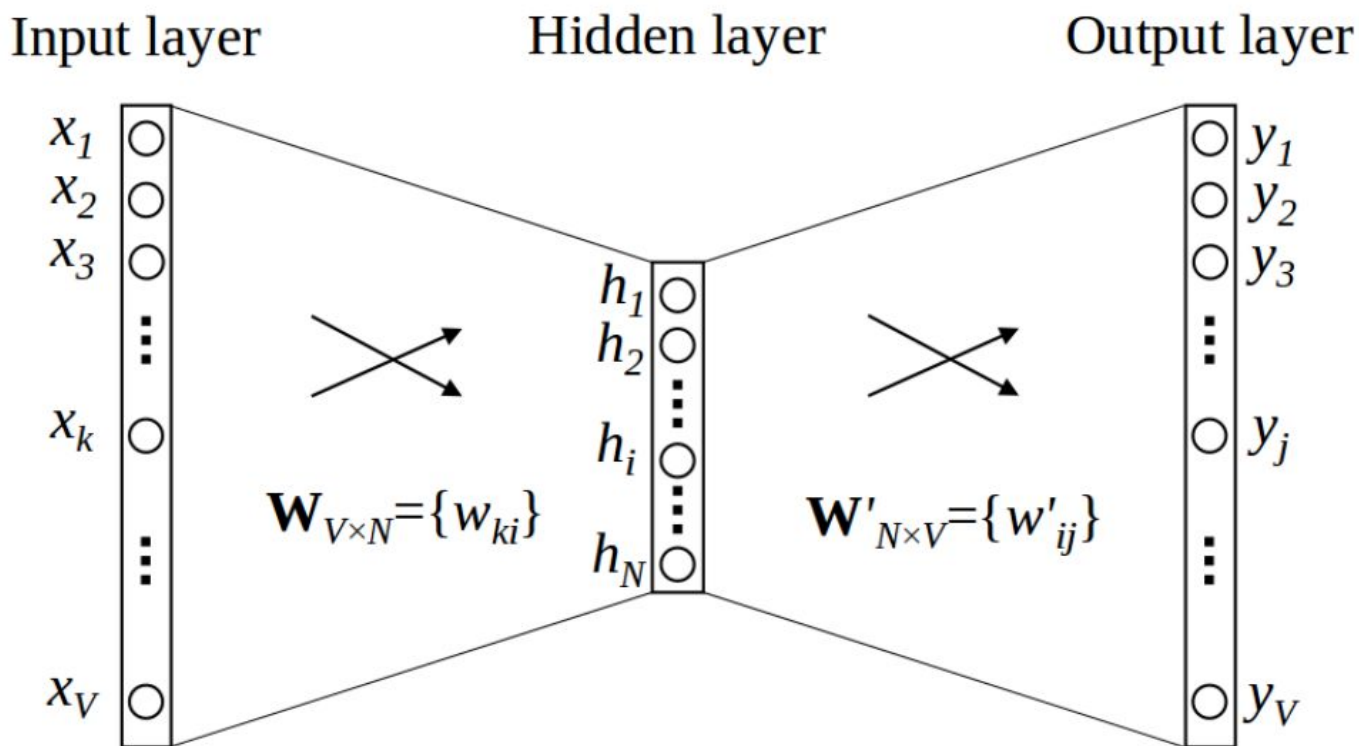
- Predicting one word each time
- Relatively fast

Skip-gram

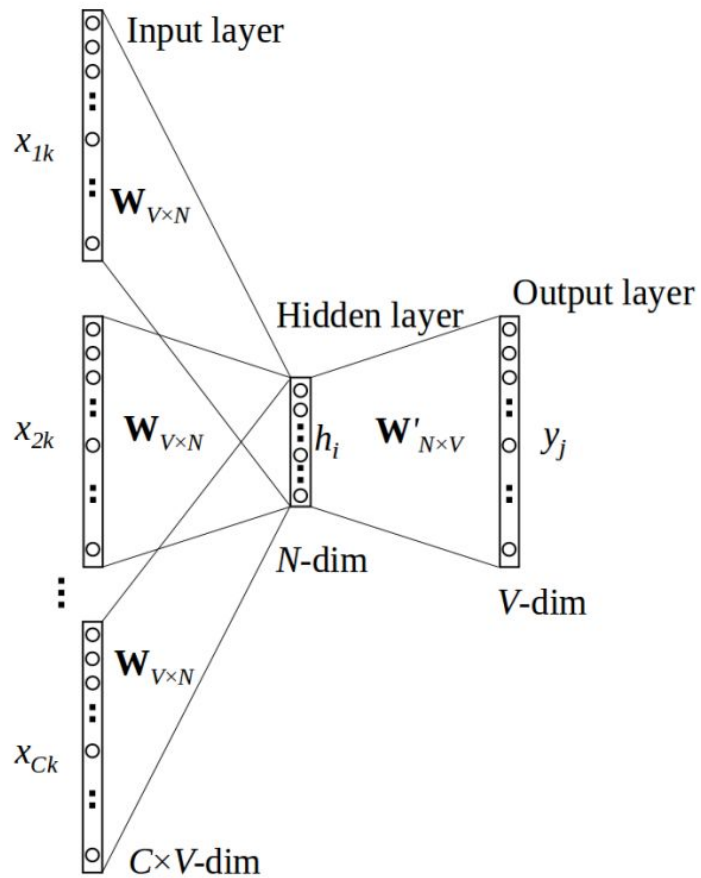
$$p(w_{i-h}, \dots, w_{i+h} | w_i)$$

Predict context ("outside")
words (position independent)
given center word

- Predicting context by one word
- Much slower
- Better with infrequent words



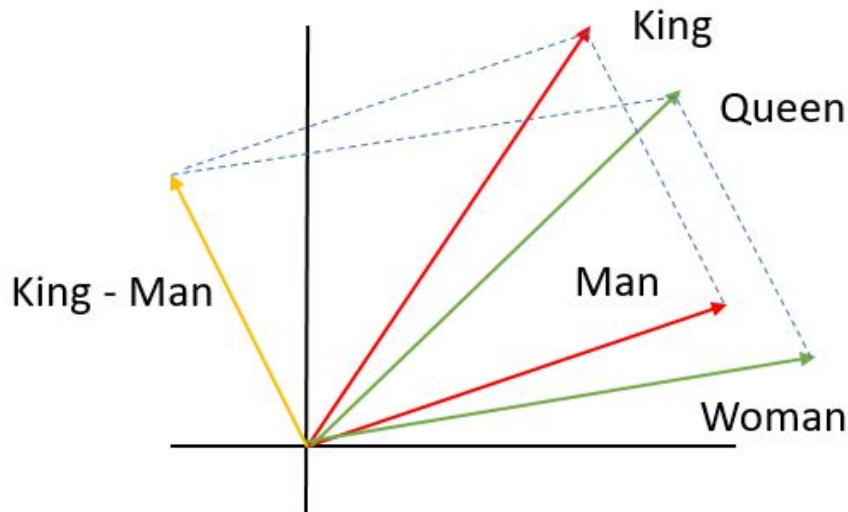
Skip-gram

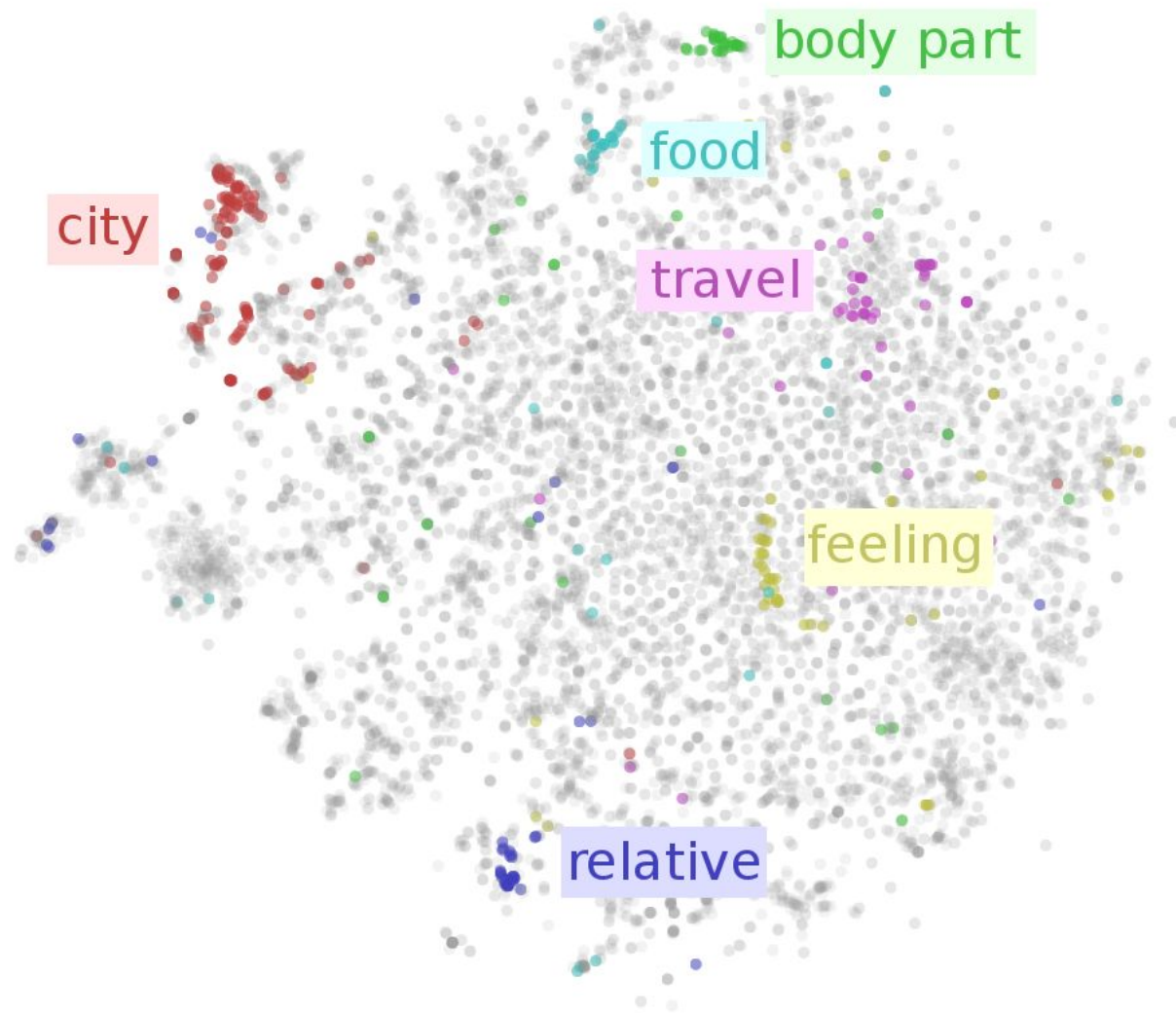


Word2vec: word analogies

King - man + woman = queen
↓ ↓ ↓ ↓
 x y y' $target$

$\cos(x - y + y', target) \rightarrow \max_{target}$





Word vectors are simply vectors of numbers that represent the meaning of a word

Approaches:

- One-hot encoding
- Bag-of-words models
- Counts of word / context co-occurrences
- TF-IDF
- Predictions of context given word (skip-gram neural network models, e.g. word2vec)