Deep Learning for NLP

- Oliver Guhr
  I833 SS2019

How is the pace of the lectures so far?

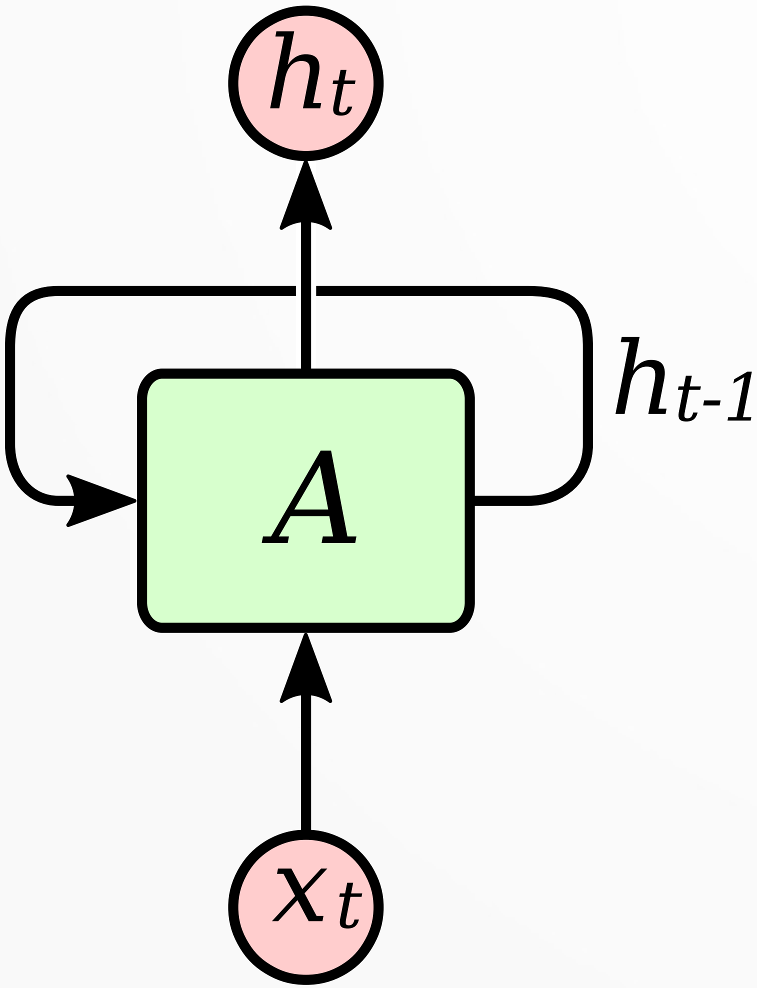
too slow / too fast / just right

a brief recap of the last lecture

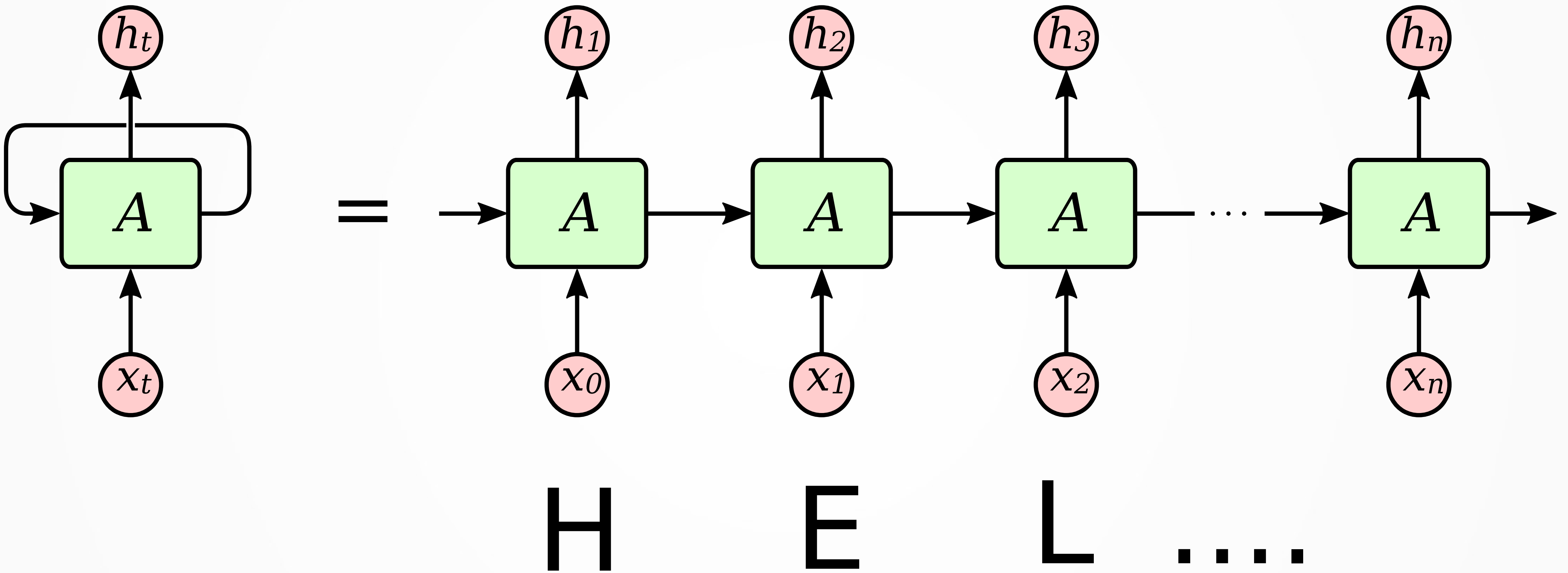
NLP Tasks

- Easy
  - Spell Checking
  - Keyword Search
- Medium
  - Parsing information from unstructured text
- Hard
  - Machine Translation
  - Semantic Analysis
  - Question Answering

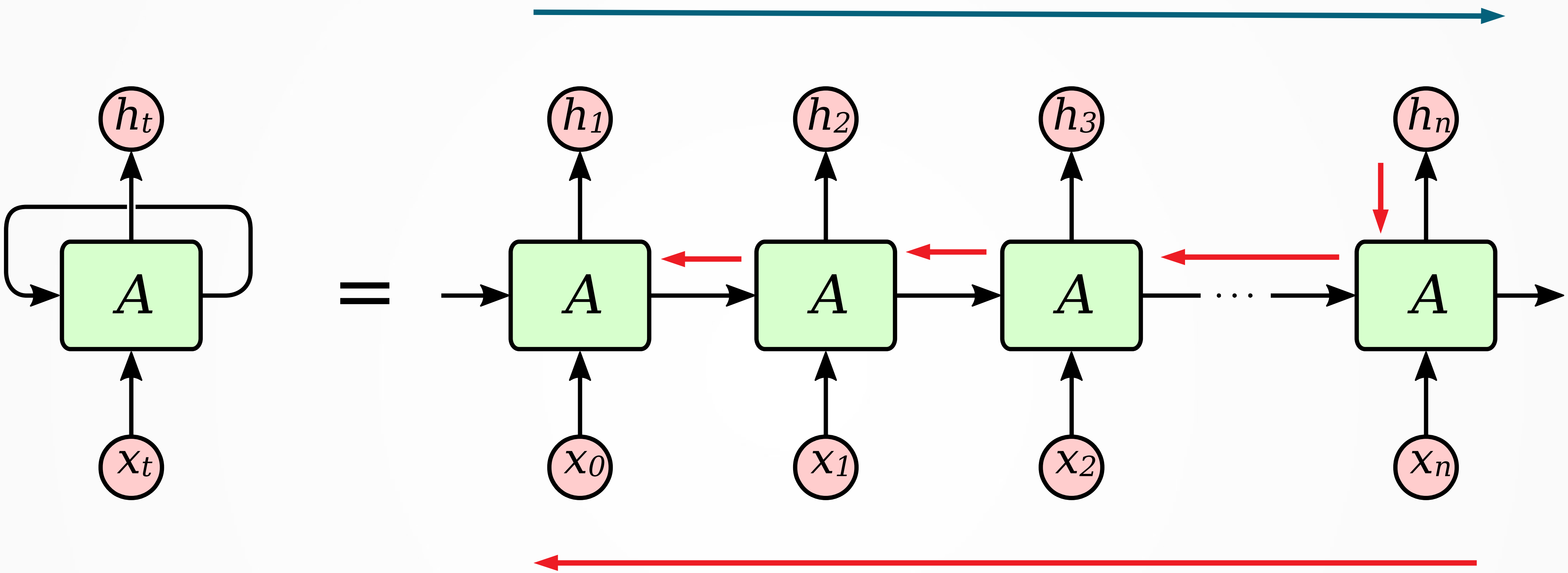
Recurrent Neural Network



$$\underbrace{h_t}_{\text{new state}} = \underbrace{A}_{\text{network function}}\left(\underbrace{h_{t-1}}_{\text{previous state}}, \underbrace{x_t}_{\text{input vector}}\right)$$

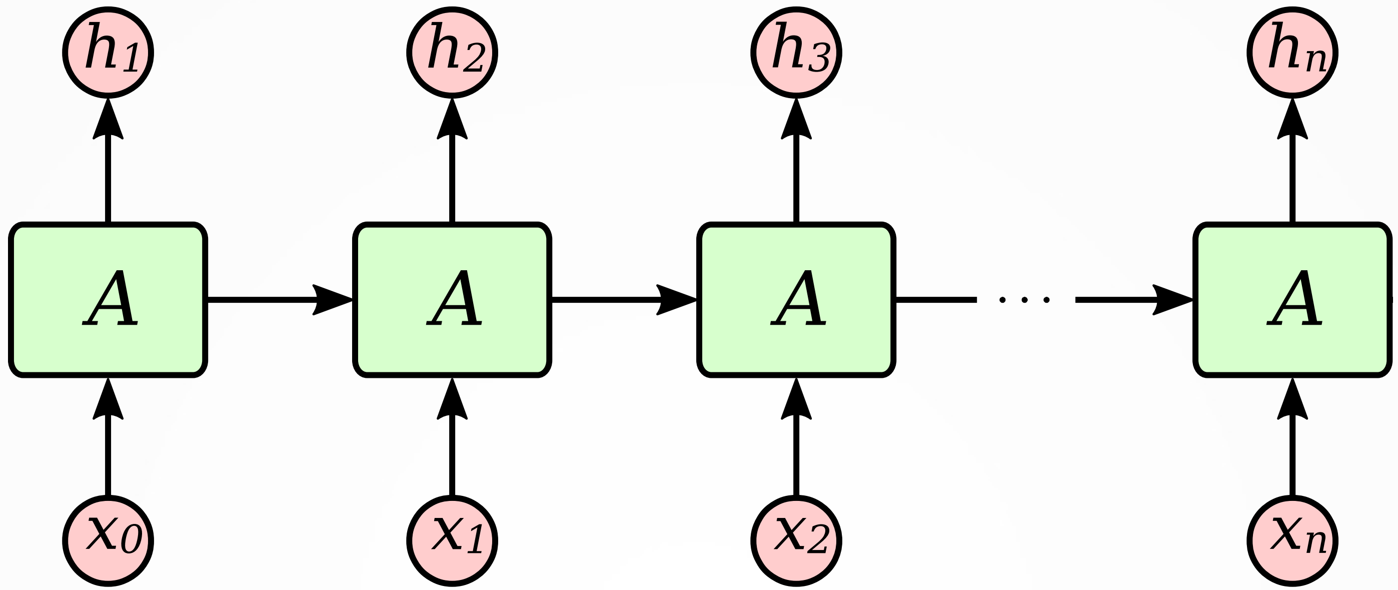

h_t
A
x_t
=
h_1 h_2 h_3 h_n
A A A A
x_0 x_1 x_2 x_n
H
E
L
....

1. forward-propagate the inputs over the unfolded network



2. back-propagate the error, back across the unfolded network

3. sum the weight changes and update all weights



$$h_1 = \tanh\left(W_{hh} h_0 + W_{xh} x_1\right)$$

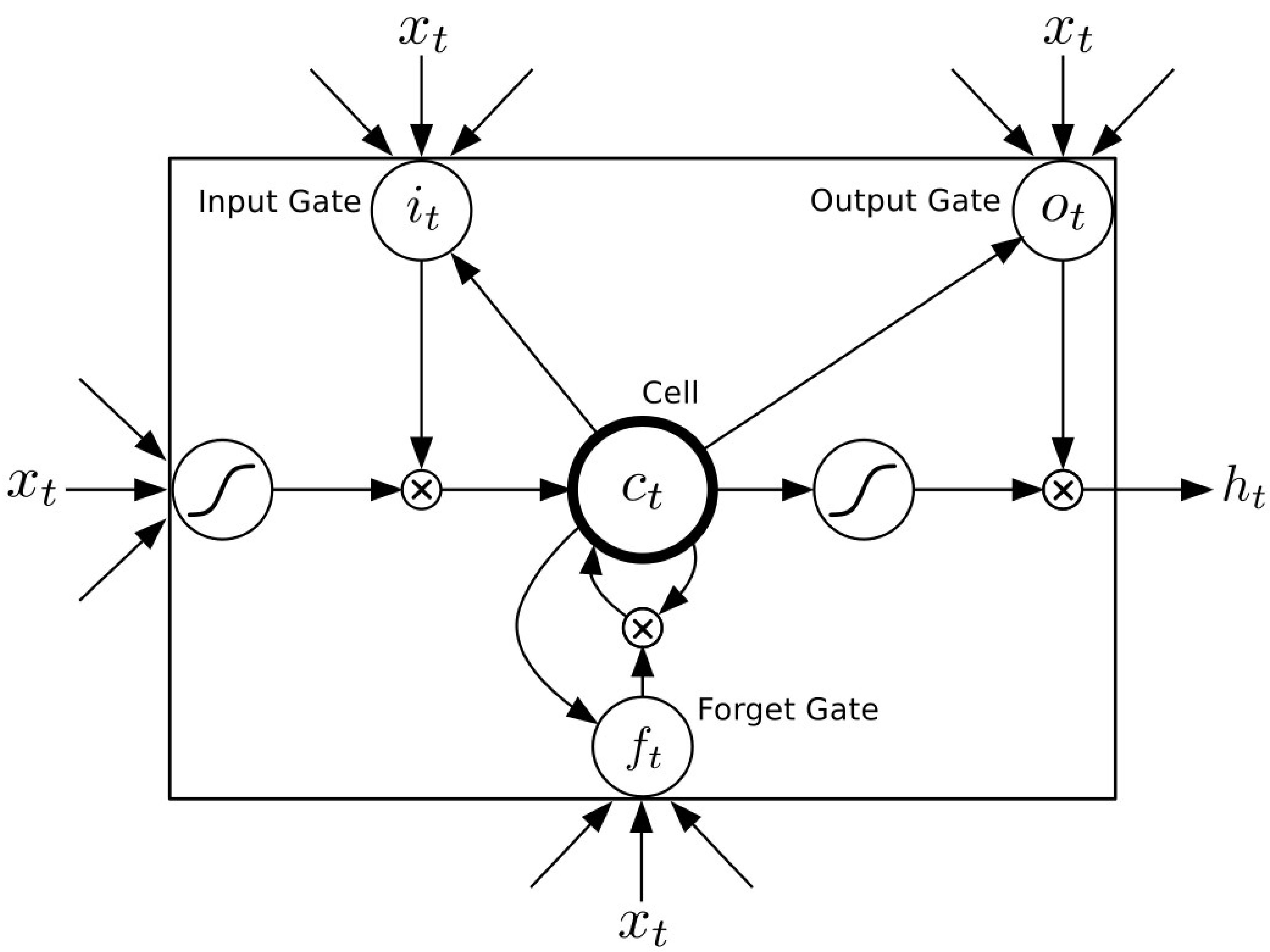
$$h_2 = \tanh\left(W_{hh}\left(\tanh\left(W_{hh} h_0 + W_{xh} x_1\right)\right) + W_{xh} x_2\right)$$

$$h_3 = \tanh\left(W_{hh}\left(\tanh\left(W_{hh}\left(\tanh\left(W_{hh} h_0 + W_{xh} x_1\right)\right) + W_{xh} x_2\right)\right) + W_{xh} x_3\right)$$

$$h_4 = \underline{\tanh\left(W_{hh}\left(\tanh\left(W_{hh}\left(\tanh\left(W_{hh}\left(\tanh\left(W_{hh}\right.\right.\right.\right.\right.\right.\right.} h_0 + W_{xh} x_1))+W_{xh} x_2))+W_{xh} x$$

Backpropagating this recursive function leads to exploding or vanishing gradients.

Single LSTM Cell



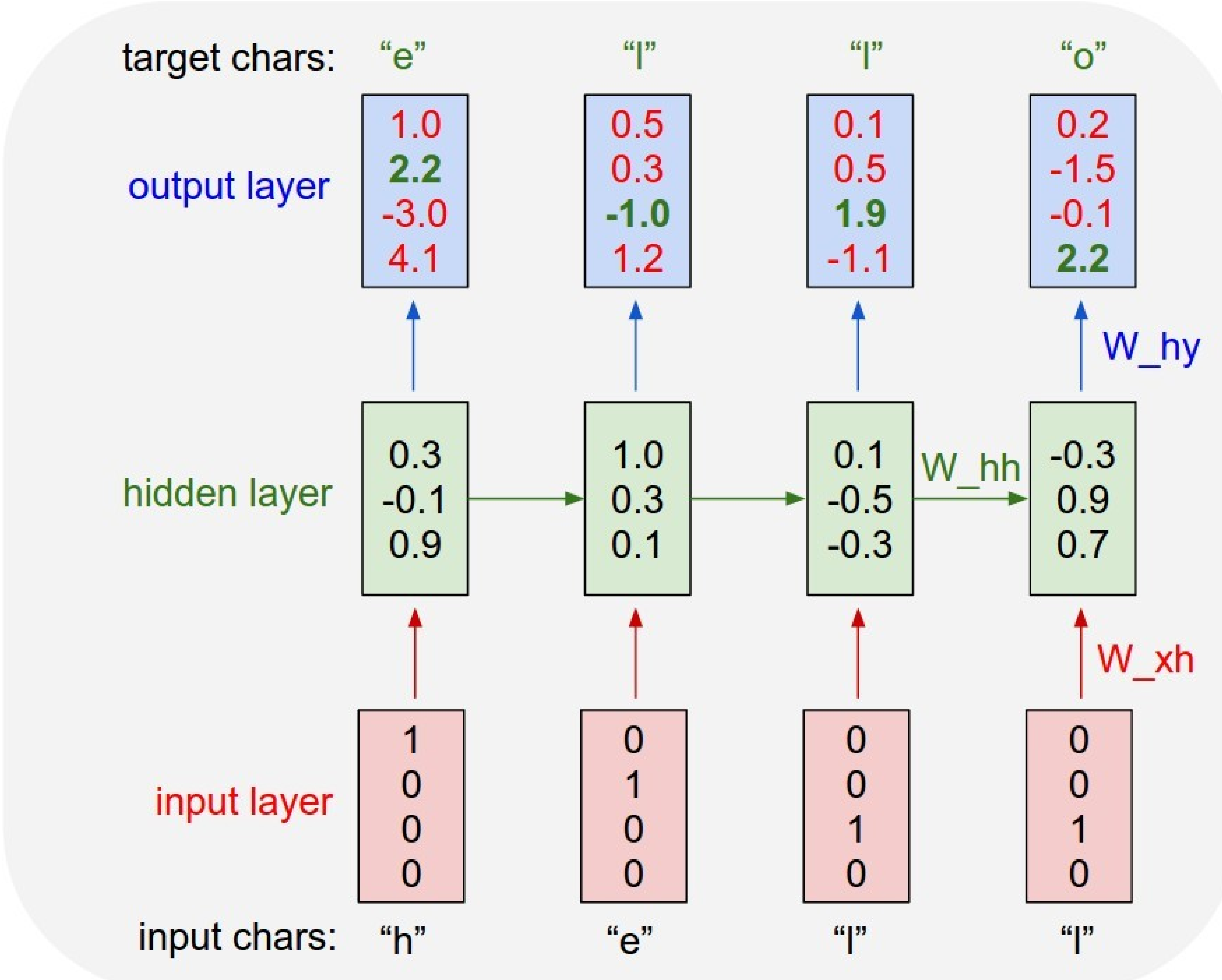
The three gates control the information flow of the cell.

Graves et al. 2013, Speech Recognition with Deep Recurrent Neural Networks

The topic for today:

Word Representations

Last time we used the one-hot encoding



Andrej Karpathy http://karpathy.github.io/2015/05/21/rnn-effectiveness/

One Hot Encoding

h	0	0	0	1
e	0	0	1	0
l	0	1	0	0
o	1	0	0	0

$$v^h = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad v^e = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \cdots \quad \longrightarrow \quad V^{hello} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Encoding

Instead of encoding single characters

h	0	0	0	1
e	0	0	1	0
l	0	1	0	0
o	1	0	0	0

You can also encode words, this is also called „Bag-Of-Words (BOW)“

hello	0	0	0	1
my	0	0	1	0
name	0	1	0	0
is	1	0	0	0

Bag-Of-Words (BOW)

- You can‘t encode words that are not in your vocabulary.
- Size of the matrix is n x n, where n is the size of your vocabulary
- The German language has an estimated number of 5,3 million words[1]. We can‘t handle such matrices.
- Since they are sparse matrices most of the entries will be zero. (inefficient)

[1] Wolfgang Klein, Page 34 http://pubman.mpdl.mpg.de/pubman/item/escidoc:1850493:4/component/escidoc: 1850492/ReichtumundArmut.pdf

Encoding

To create a dense vector representation for the words, we could use the idea that words are related to each other.

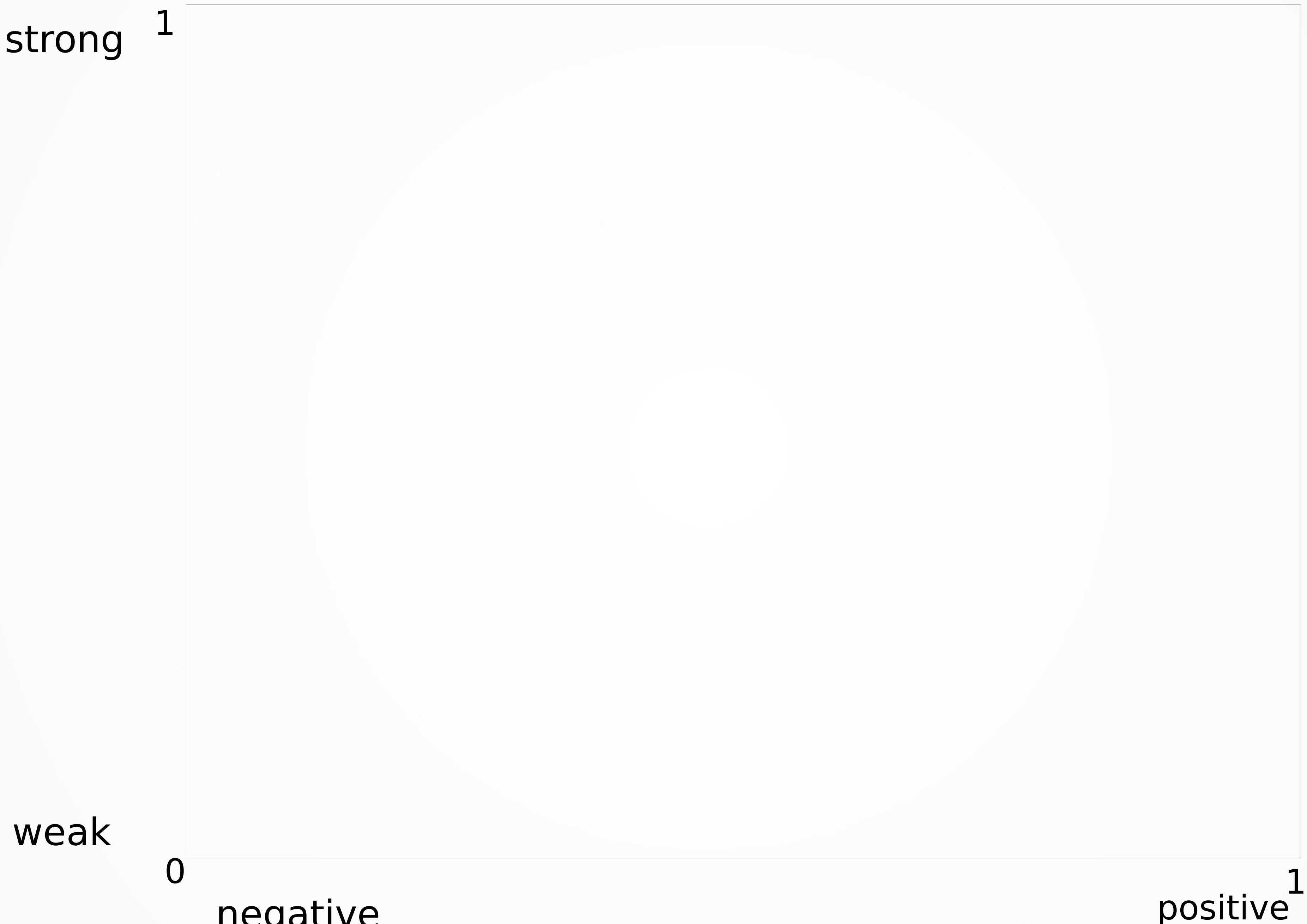
For example: nice and good

Vectors and Vector Space

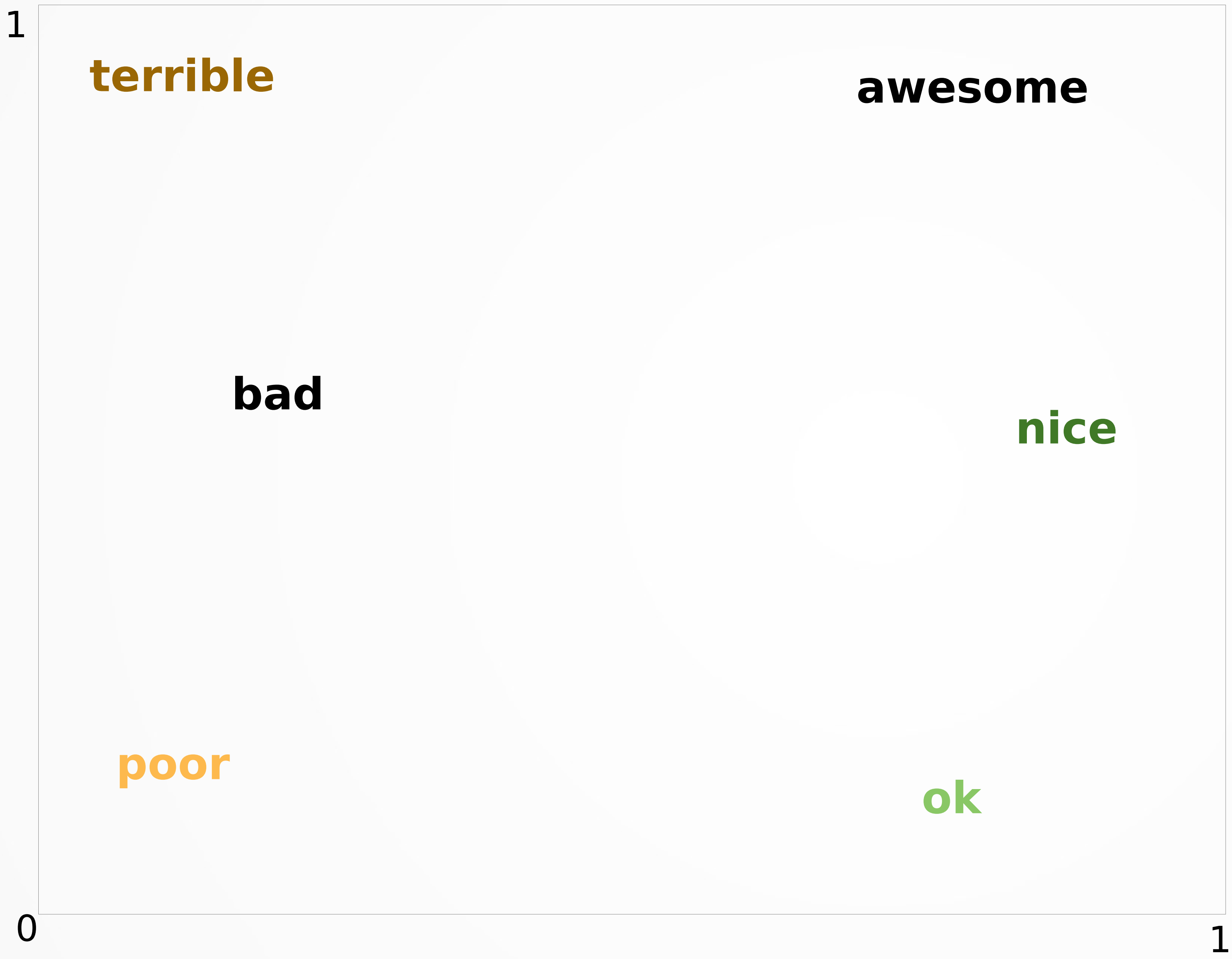
- A sequence of numbers that is used to identify a point in space is called a vector.

- A list of vectors that belong to the same data set, is called a vector space.

How can we encode words using dense vectors?


strong
1
weak
0
negative
positive
1


strong
1
terrible
awesome
bad
nice
poor
ok
weak
0
negative
1
positive



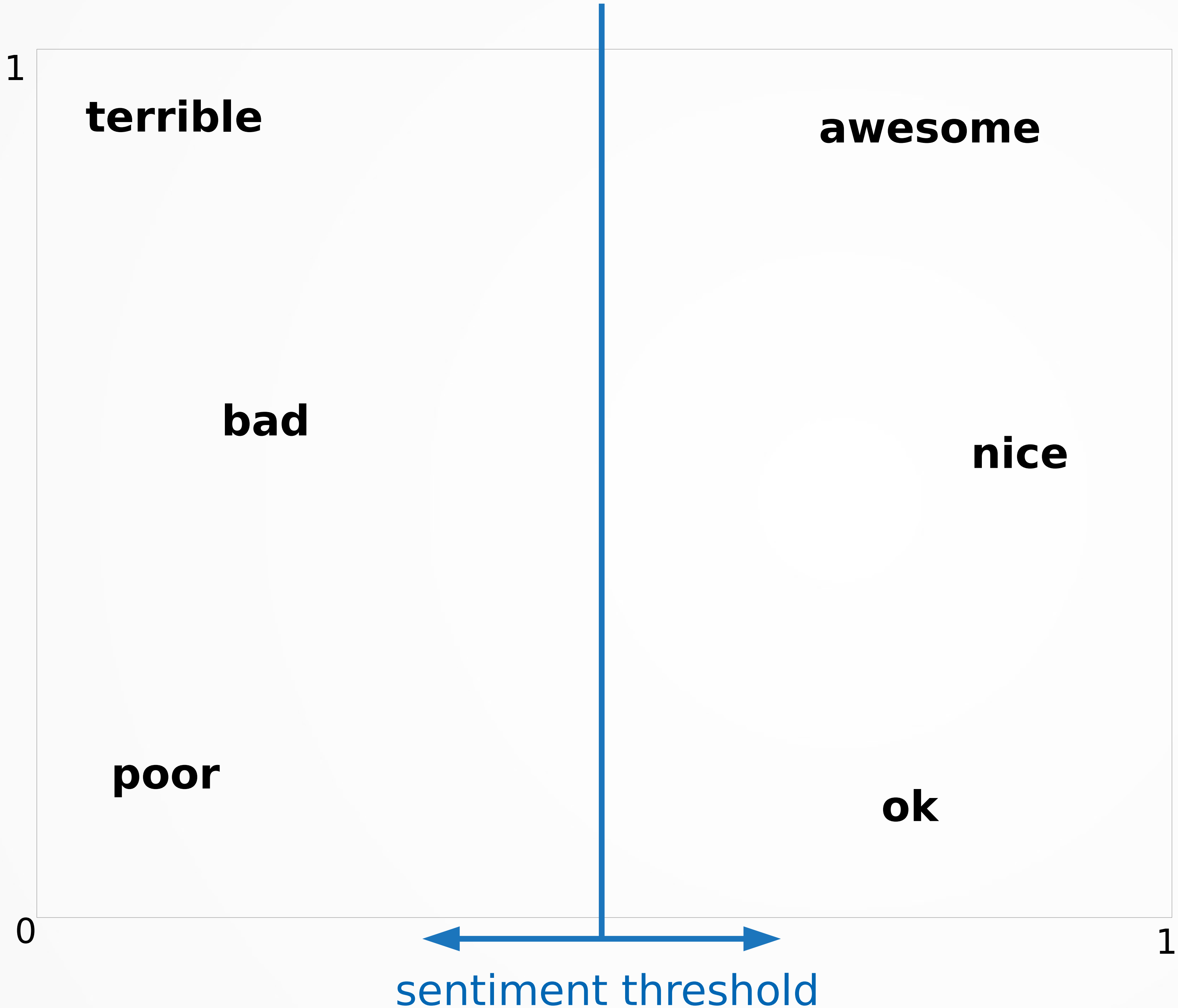
The coordinates on our map can be used as vector representations of our words.

$$v_{ok} = [0.75, 0.15]$$

$$v_{nice} = [0.85, 0.50]$$

$$v_{poor} = [0.15, 0.18]$$

$$v_{terrible} = [0.10, 0.91]$$



Now a network can distinguish between positive and negative words by learning a threshold.

$$v_{ok} = [0.75, 0.15]$$

$$v_{nice} = [0.85, 0.50]$$

$$v_{poor} = [0.15, 0.18]$$

$$v_{terrible} = [0.10, 0.91]$$

Distance and similarity

Since our words are now vectors, we can use the euclidean distance to calculate similarity of two words.

$$\| v_{nice} - v_{ok} \| = 0.364$$

$$\| v_{terrible} - v_{ok} \| = 1$$

How do we train word vectors?

Supervised and unsupervised learning

Supervised Learning

Supervised Learning

- These vectors are also called embeddings
- An embedding is n-dimensional vector
- It will serve as an input layer an will be trained along with your model

“Hello” = [0.6614, 0.2669, 0.6213, -0.4519]

Unsupervised Learning

Distributional Hypothesis

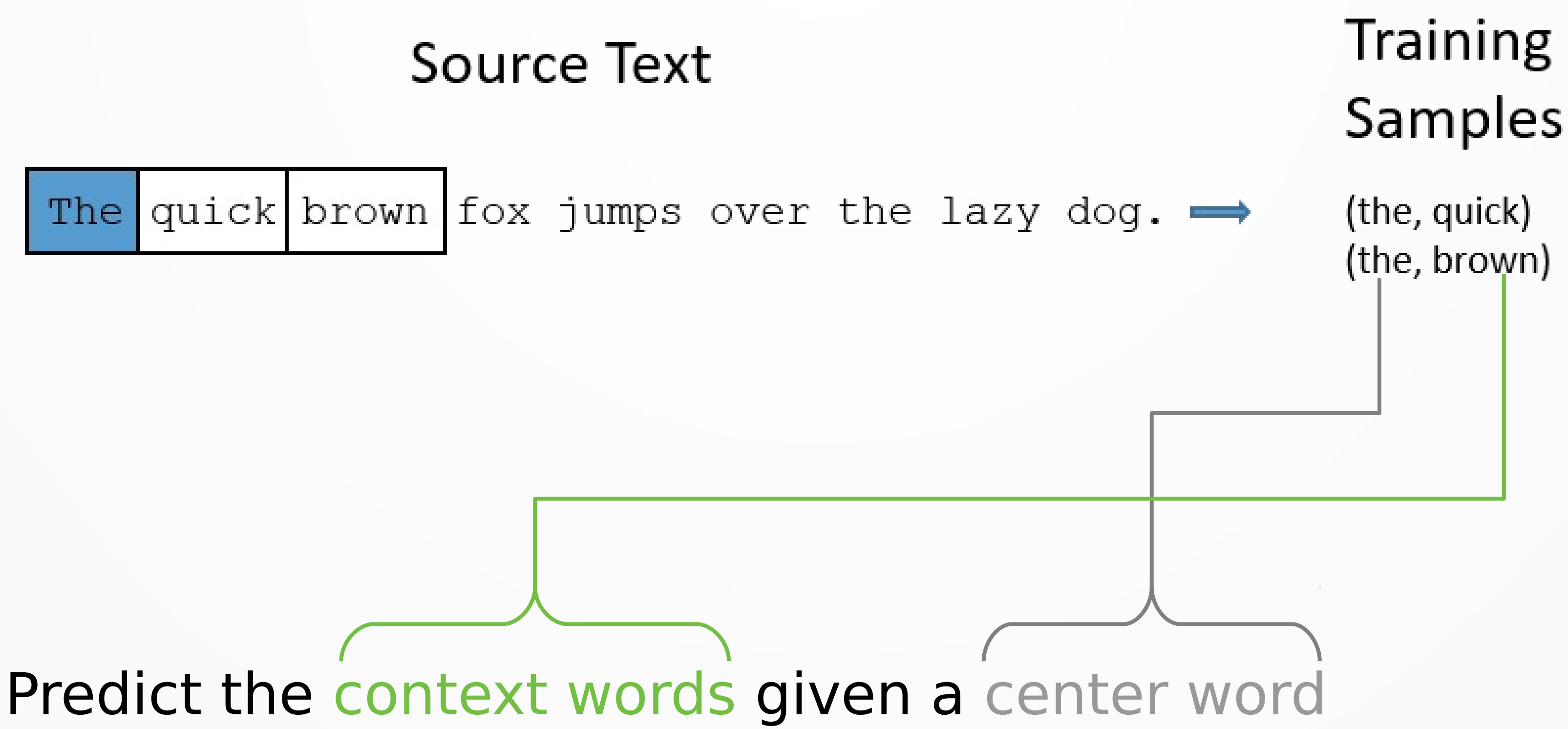
Words that occur in the same contexts tend to have similar meanings.
Harris (1954)

A word is characterized by the company it keeps.
Firth (1957)

Word Vector

- Skip-Gram Model (SG)
  - Predict the context words given a center word
- Continuous Bag of Words (CBOW)
  - Predict center word from a bag of unsorted context words

Skip-Gram



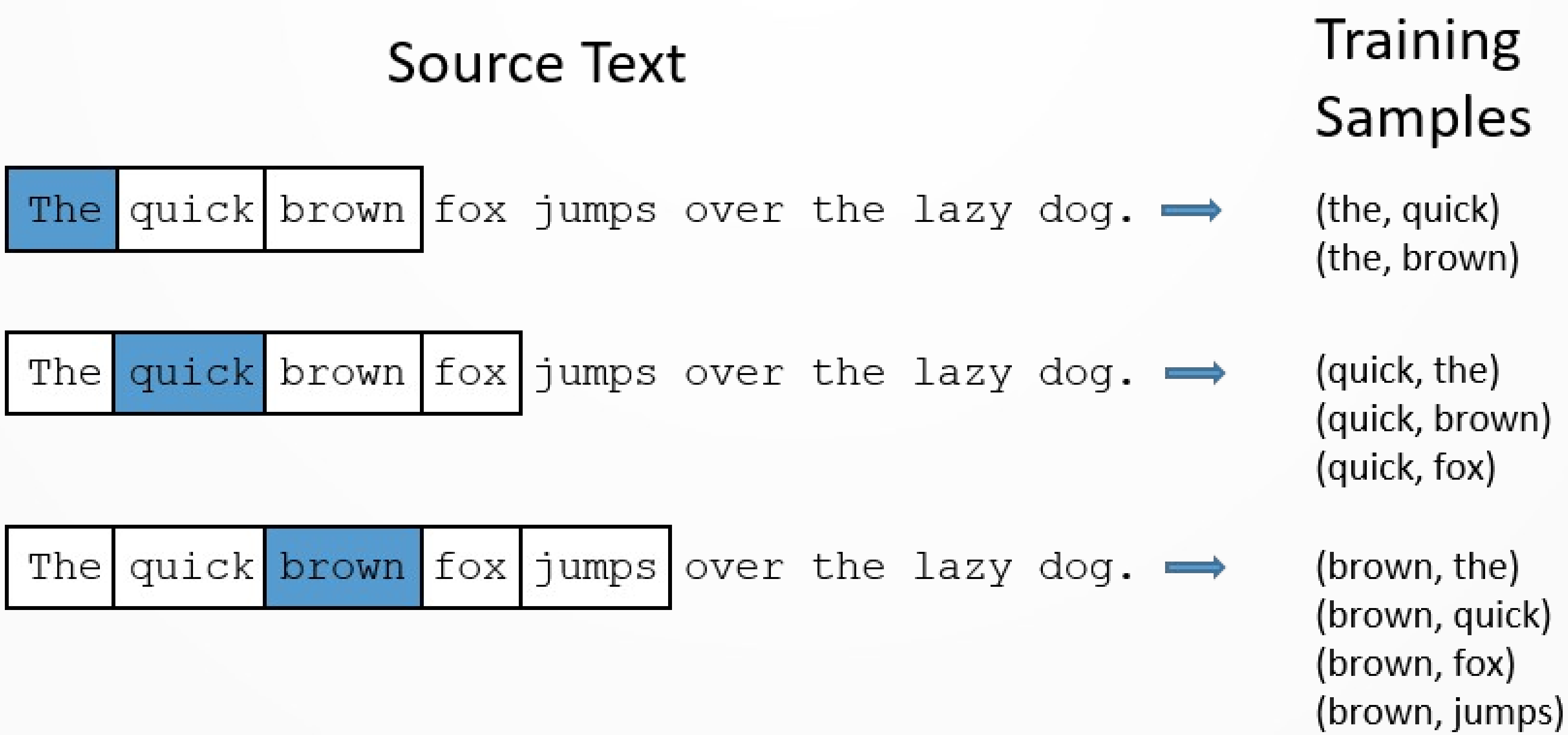
McCormick, Word2Vec Tutorial, http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Skip-Gram

Source Text	Training Samples
The quick brown fox jumps over the lazy dog.	(the, quick) (the, brown)
The quick brown fox jumps over the lazy dog.	(quick, the) (quick, brown) (quick, fox)

McCormick, Word2Vec Tutorial, http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Skip-Gram



We are using a window size of 2. Meaning: two words behind and 2 words ahead of the center word.

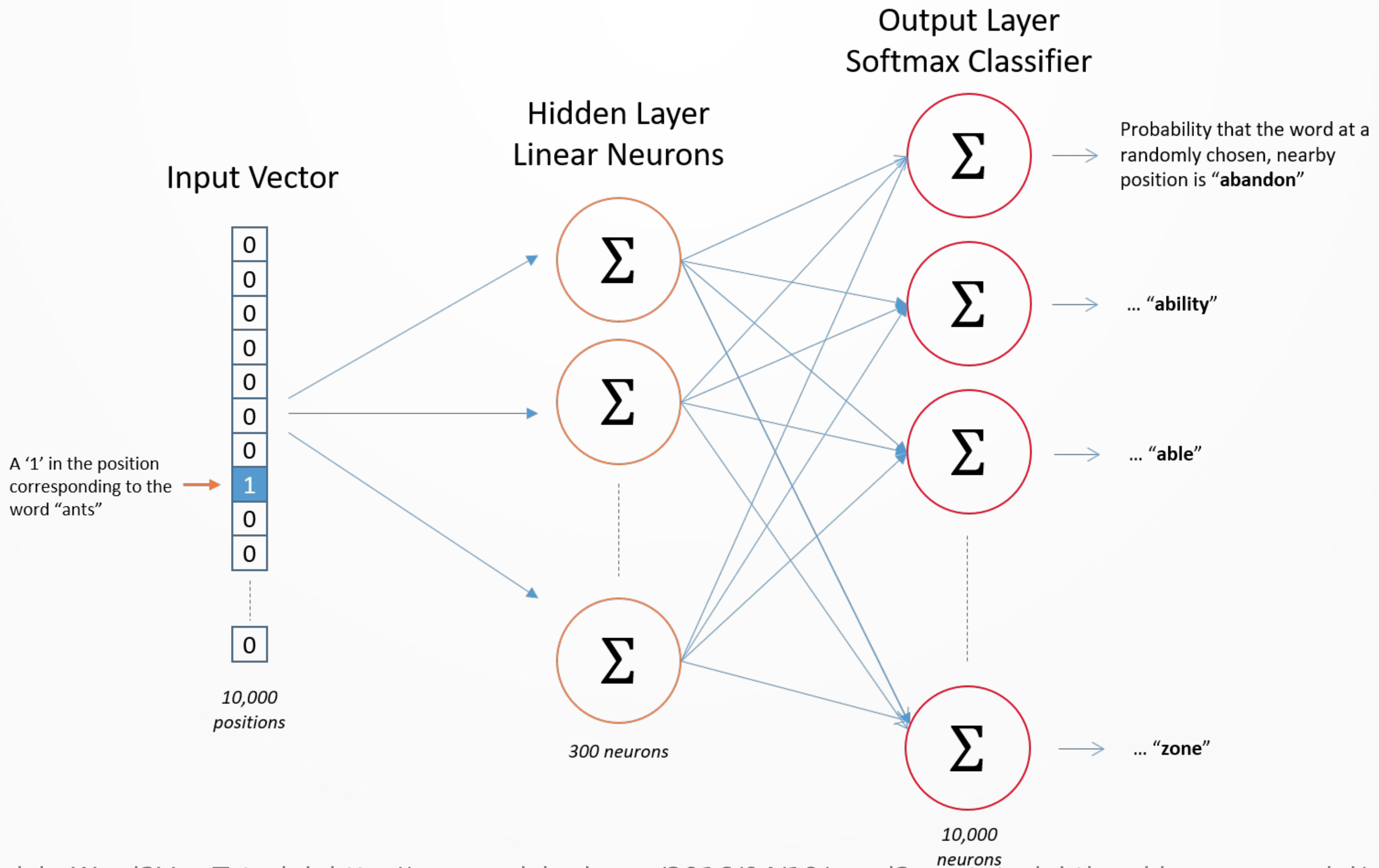
McCormick, Word2Vec Tutorial, http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Skip-Gram

Source Text	Training Samples
The quick brown fox jumps over the lazy dog. →	(the, quick) (the, brown)
The **quick** brown fox jumps over the lazy dog. →	(quick, the) (quick, brown) (quick, fox)
The quick **brown** fox jumps over the lazy dog. →	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown **fox** jumps over the lazy dog. →	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

McCormick, Word2Vec Tutorial, http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

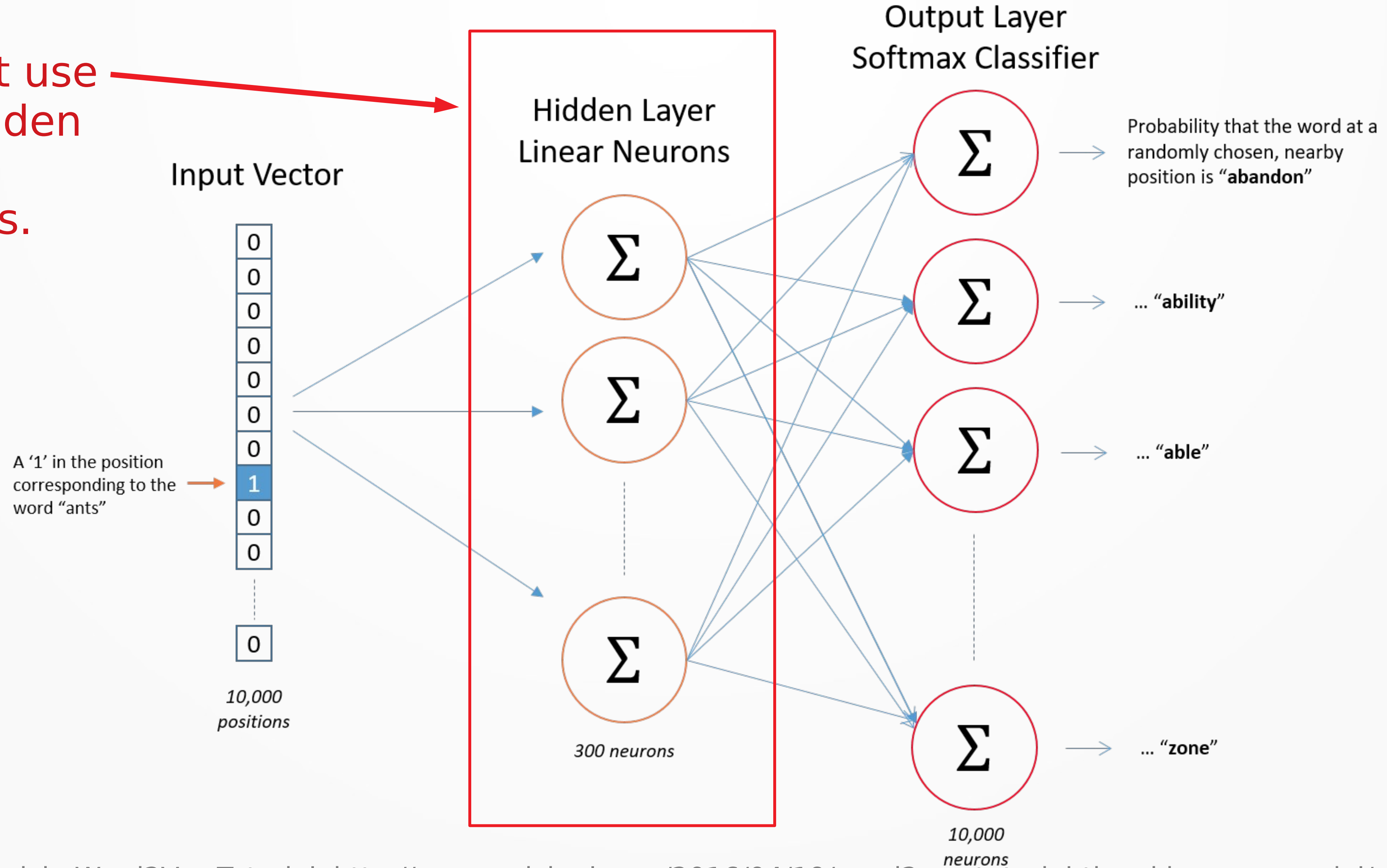
Skip-Gram



McCormick, Word2Vec Tutorial, http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

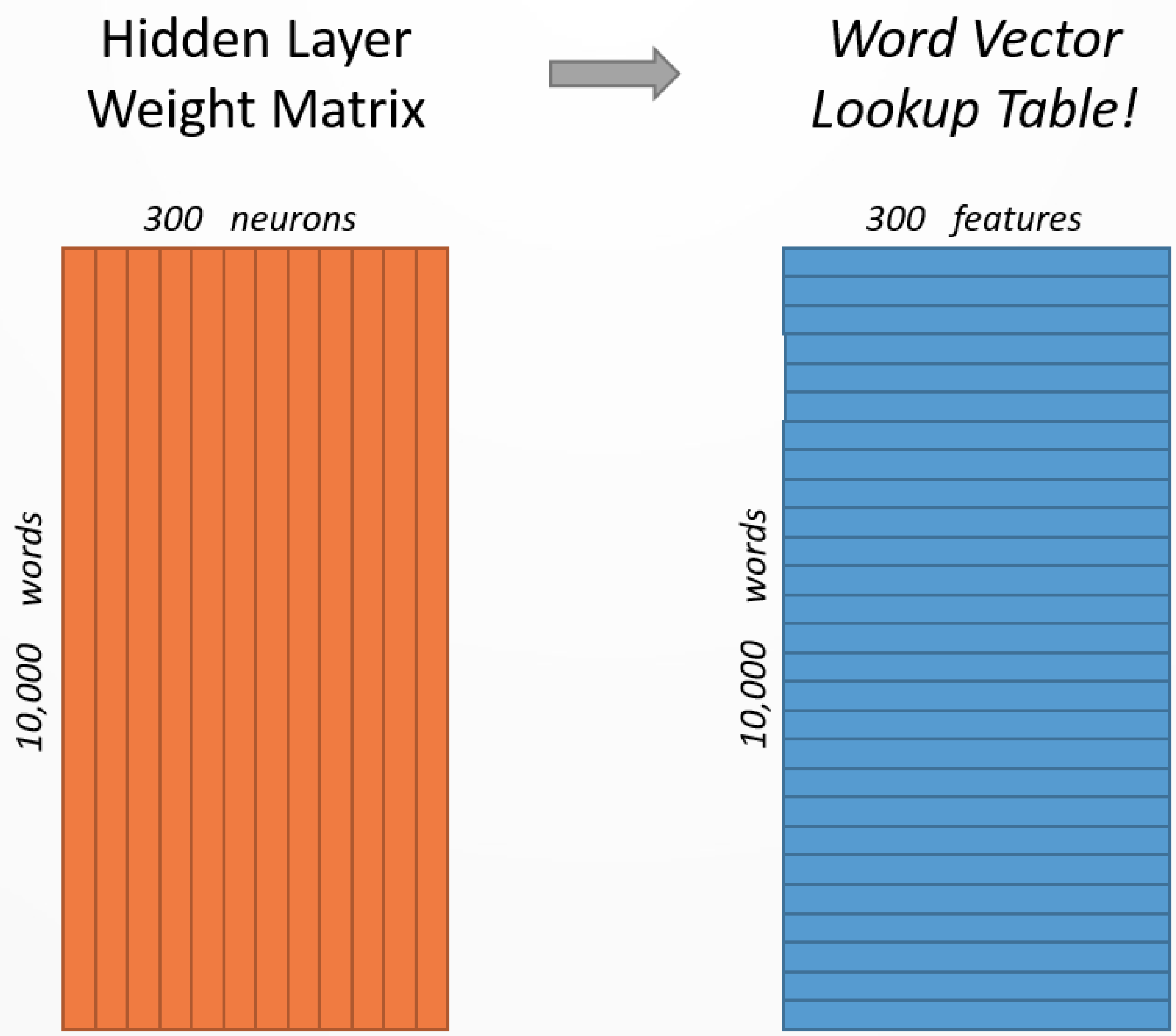
Skip-Gram

We just use the hidden layer weights.



McCormick, Word2Vec Tutorial, http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Skip-Gram



McCormick, Word2Vec Tutorial, http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Ideas:

- For words with similar contexts, our model needs to produce a similar result. This will motivate the model to learn similar weights, that we use as words vectors.

- This way we „compressed“ our 1 x 10000 sparse one-hot vector to a 1 x 300 dense vector.

- **We can now reuse this pretrained vectors in other models.**

- Read the tutorial for more details:
  - http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Papers

- Word2Vecc

  Efficient Estimation of Word Representations inVector Space Mikolov et al. 2013

- Negative Sampling (for more efficent training)

  Distributed Representations of Words and Phrasesand their Compositionality Mikolov et al. 2013

- FastText (use of subword information)

  Enriching Word Vectors with Subword Information Bojanowski, Grave, Joulin, Mikolov 2017

Wiki word vectors

We are publishing pre-trained word vectors for 294 languages, trained on *Wikipedia* using fastText. These vectors in dimension 300 were obtained using the skip-gram model described in *Bojanowski et al. (2016)* with default parameters.

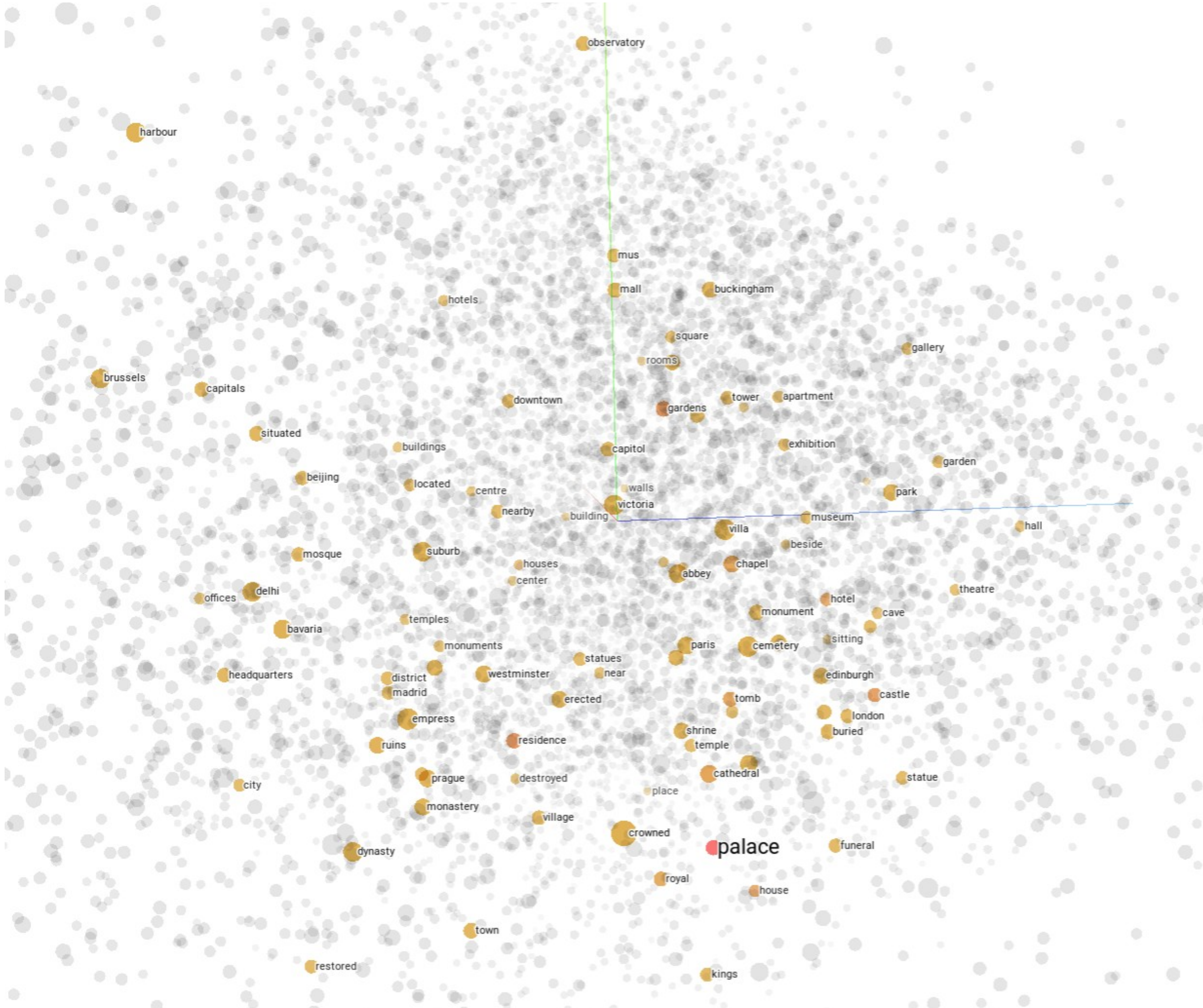
Please note that a newer version of multi-lingual word vectors are available at: Word vectors for 157 languages.

Models

The models can be downloaded from:

Abkhazian: *bin+text*, *text*	Acehnese: *bin+text*, *text*	Adyghe: *bin+text*, *text*
Afar: *bin+text*, *text*	Afrikaans: *bin+text*, *text*	Akan: *bin+text*, *text*
Albanian: *bin+text*, *text*	Alemannic: *bin+text*, *text*	Amharic: *bin+text*, *text*
Anglo_Saxon: *bin+text*, *text*	Arabic: *bin+text*, *text*	Aragonese: *bin+text*, *text*
Aramaic: *bin+text*, *text*	Armenian: *bin+text*, *text*	Aromanian: *bin+text*, *text*
Assamese: *bin+text*, *text*	Asturian: *bin+text*, *text*	Avar: *bin+text*, *text*
Aymara: *bin+text*, *text*	Azerbaijani: *bin+text*, *text*	Bambara: *bin+text*, *text*
Banjar: *bin+text*, *text*	Banyumasan: *bin+text*, *text*	Bashkir: *bin+text*, *text*
Basque: *bin+text*, *text*	Bavarian: *bin+text*, *text*	Belarusian: *bin+text*, *text*
Bengali: *bin+text*, *text*	Bihari: *bin+text*, *text*	Bishnupriya Manipuri: *bin+text*, *text*
Bislama: *bin+text*, *text*	Bosnian: *bin+text*, *text*	Breton: *bin+text*, *text*
Buginese: *bin+text*, *text*	Bulgarian: *bin+text*, *text*	Burmese: *bin+text*, *text*
Buryat: *bin+text*, *text*	Cantonese: *bin+text*, *text*	Catalan: *bin+text*, *text*
Cebuano: *bin+text*, *text*	Central Bicolano: *bin+text*, *text*	Chamorro: *bin+text*, *text*
Chavacano: *bin+text*, *text*	Chechen: *bin+text*, *text*	Cherokee: *bin+text*, *text*

exploring the vector space


observatory
harbour
mus
mall
buckingham
hotels
square
rooms
gallery
brussels
capitals
downtown
tower
apartment
gardens
situated
buildings
capitol
exhibition
garden
beijing
located
centre
walls
park
nearby
victoria
building
museum
hall
villa
mosque
suburb
beside
houses
chapel
abbey
center
delhi
offices
theatre
hotel
monument
cave
temples
bavaria
sitting
monuments
paris
cemetery
statues
headquarters
district
westminster
near
edinburgh
madrid
erected
tomb
castle
empress
london
shrine
residence
buried
ruins
temple
cathedral
statue
city
prague
destroyed
place
monastery
village
crowned
palace
funeral
dynasty
royal
house
town
restored
kings

What did the model learn?

- Idea:
- reduce demensions with PCA
- create a 3D map of the vector space:
- https://projector.tensorflow.org/

Preprocessing

Preprocessing

- Format you text to be predictable and analyzable
- It often has a significant impact on the performance
- Depending on the domain and your model different steps may be required
- For example:
  - Cleaning not useful characters and word
  - Transform words into a standardized form
  - Clipping your data to equal length

Preprocessing

Lower casing:

Berlin, berlin, berLin, BERLIN → berlin

Preprocessing

cleaning:

I ate sup at @starbugs → i ate sup at starbugs

#super-nice → super nice

https://some.urlhttps://google.de is super → is super

Preprocessing

Normalization:

otw → on the way

:) :-) → happy-smile

:(:-(;-(→ sad-smile

Preprocessing

Noise Removal:

...berlin..

Berlin! → berlin

<i>Berlin</i>

Preprocessing

concatenate named entities:

New York	→	new_york
“Die Zeit”	→	die_zeit

Preprocessing

- Other ideas: check spelling, remove numbers, etc.
- Not all preprocessing step will improve your results. Measure the effect!
- Further information:
  - https://mlwhiz.com/blog/2019/01/17/deeplearning_nlp_preprocess/
  - https://www.kaggle.com/saxinou/nlp-01-preprocessing-data

Dataset Augmentation

Dataset Augmentation

Create artificial data and add it to your dataset

„Hello my name is Anna“

„Hello my name is John“

„Hello my name is Ali“

„Hello my name is Zoe“

Dataset Augmentation

Transform the data, but keep the class:

Text „*You are smart*“ class flattery

Synonyms „clever“, „bright“, „brainy“

You are clever

You are bright

You are brainy

Dataset Augmentation

- More Ideas:
  - Random insertion / swap / deletion of words
  - Split longer texts into sentences, recombine these into new texts.
  - Translate in a different language and translate back
  - Be creative :)

Dataset Augmentation

- Dataset Augmentation is a regularization technique
- The goal is to improve the robustness and generalization of your model

Dataset Augmentation

- Test your model before and after you applied the augmentation.
- Be careful to not apply transformations that change the class.
- When you compare different algorithms, compare them usign the same augmentations.

„If you can't measure it, you can't improve it.“

-Peter Drucker

How to optimize:

1) create a train/test split
2) Train your model (start with a simple model!)
3) measure its performance
4) optimize your model
5) Go to 2 :)

Scores: Accuracy

- $$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

- $$\text{Accuracy} = \frac{\text{number correctly predicted samples}}{\text{total number of samples}}$$

Scores: Accuracy

- Does not work well if your classes are not balanced.

- Example:
  - Test on: 990 negative samples and 10 postive samples
  - A model that classifies all samples as negative will achieve a accuracy of 99%.

- <u>Do not just trust in accuracy scores</u>

Scores: F1

$$precision = \frac{true\ \ positives}{true\ \ positives + false\ \ positives}$$

$$recall = \frac{true\ \ positives}{true\ \ positives + false\ \ negatives}$$

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Identify offensive language

using pretrained vectors and FastText

Assignment

- shared task on the identification of offensive language from GermEval 2018
- Project Page
  - https://projects.fzai.h-da.de/iggsa/projekt/
- Dataset
  - https://github.com/uds-lsv/GermEval-2018-Data

Binary Classification Task

- The task is to decide whether a message includes some form of offensive language or or not.

- OFFENSE
  - Juhu, das morgige Wetter passt zum Tag SCHEIßWETTER
  - @KarlLagerfeld ist in meinen Augen strunzdumm wie ein Knäckebrot.

- OTHER
  - @Sakoelabo @Padit1337 @SawsanChebli Nicht alle Staatssekretäre kann man ernst nehmen.
  - Die Türkei führt einen Angriffskrieg und die @spdde inkl. @sigmargabriel rüstet noch ihre Panzer auf.

Your Task

- Build a classifyer usign fasttext that can solve subtask decide if a mesage contains offense language or not.

- The top team from TU Wien scored
  - Accuracy 79,53%
  - F1 76,77%

Hints

- Project Page GermEval
  - https://projects.fzai.h-da.de/iggsa/projekt/
- Dataset GermEval 2018
  - https://github.com/uds-lsv/GermEval-2018-Data
- Word Vectors and Infos
  - https://fasttext.cc/
- Github Repo
  - https://github.com/facebookresearch/fastText
- Look out for some samples

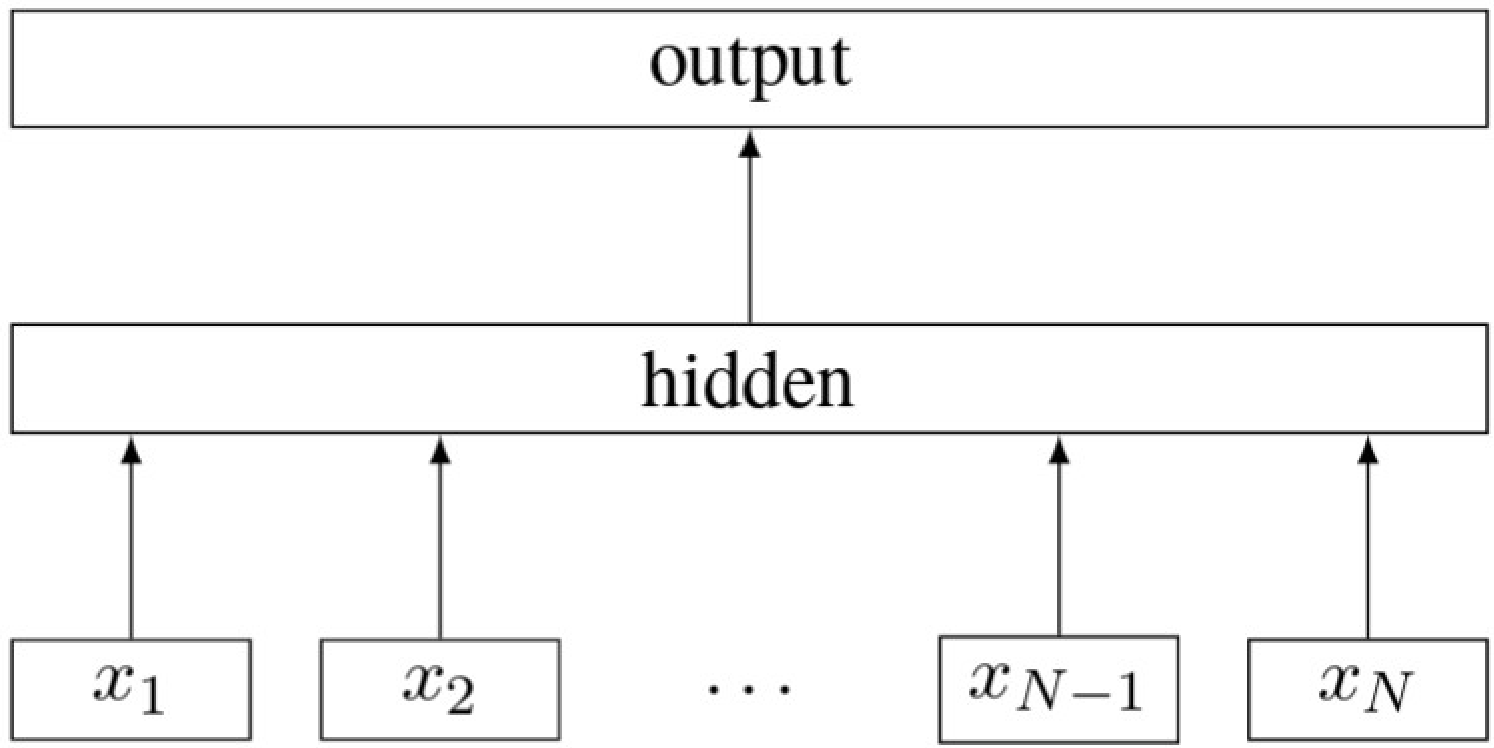


Figure 1: Model architecture of `fastText` for a sentence with N ngram features $x_1, \ldots, x_N$. The features are embedded and averaged to form the hidden variable.