Natural Language Processing II

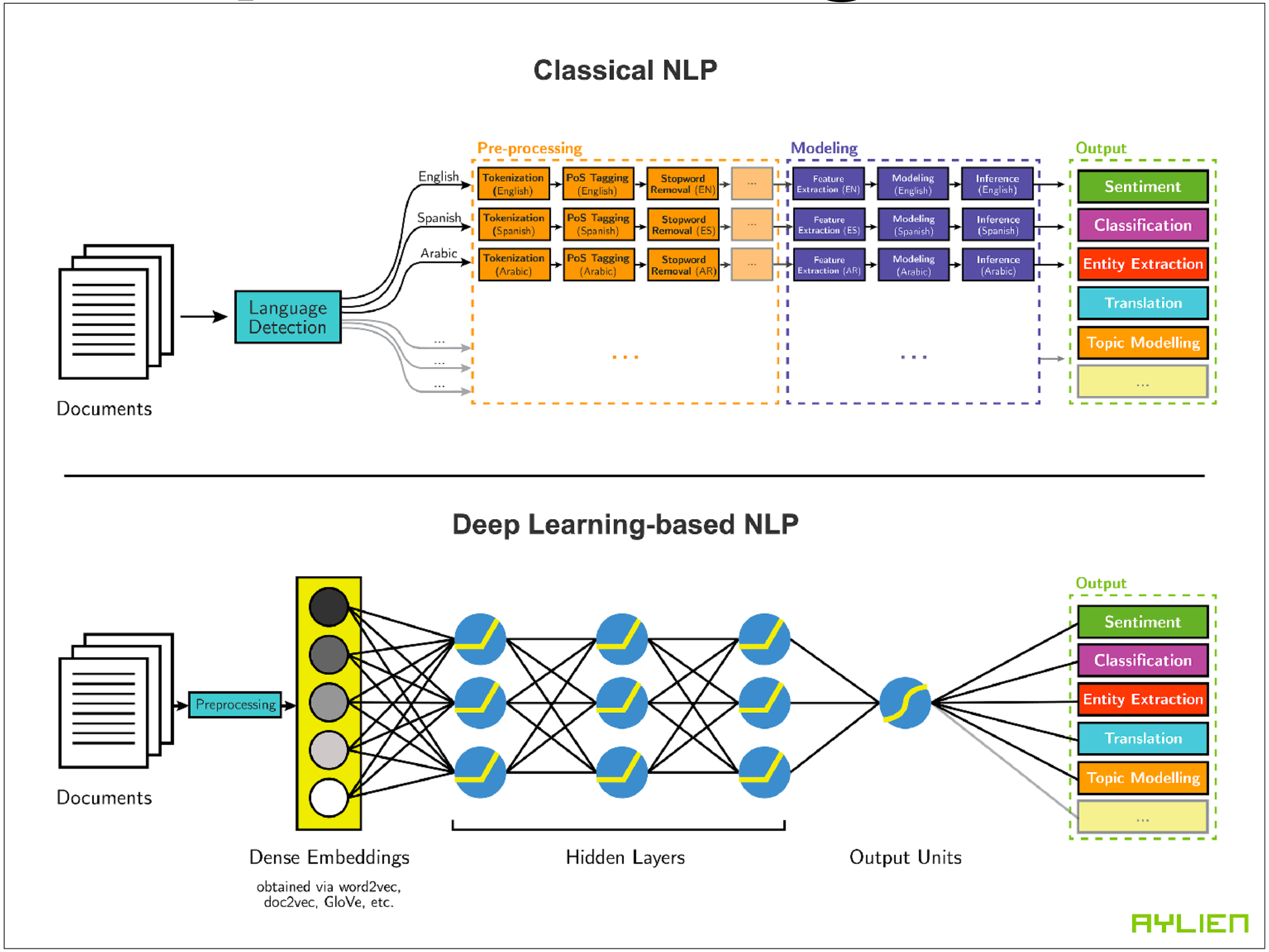
Jiří Materna

Machine Learning College

Outline

- Preprocessing for deep learning in NLP
- Recurrent neural networks
- Word embeddings and word2vec
- The Skip-gram model
- Text classification with word embeddings
- Subword tokenization
- LSTM and GRU
- Attention is all you need
- Transformers (GPT3, BERT, XLNET)
- Practical task on classification using BERT

Deep Learning in NLP



Source: https://blog.aylien.com/

Encoding and Unicode

ASCII

```
H  e  l  l  o

48 65 6c 6c 6f
```

Unicode

```
   H        e        l        l        o        ☺

00000048 00000065 0000006c 0000006c 0000006f 0000263a
```

Encoding and Unicode

UTF-8

```
H  e  l  l  o  ☺

48 65 6c 6c 6f e298ba
```

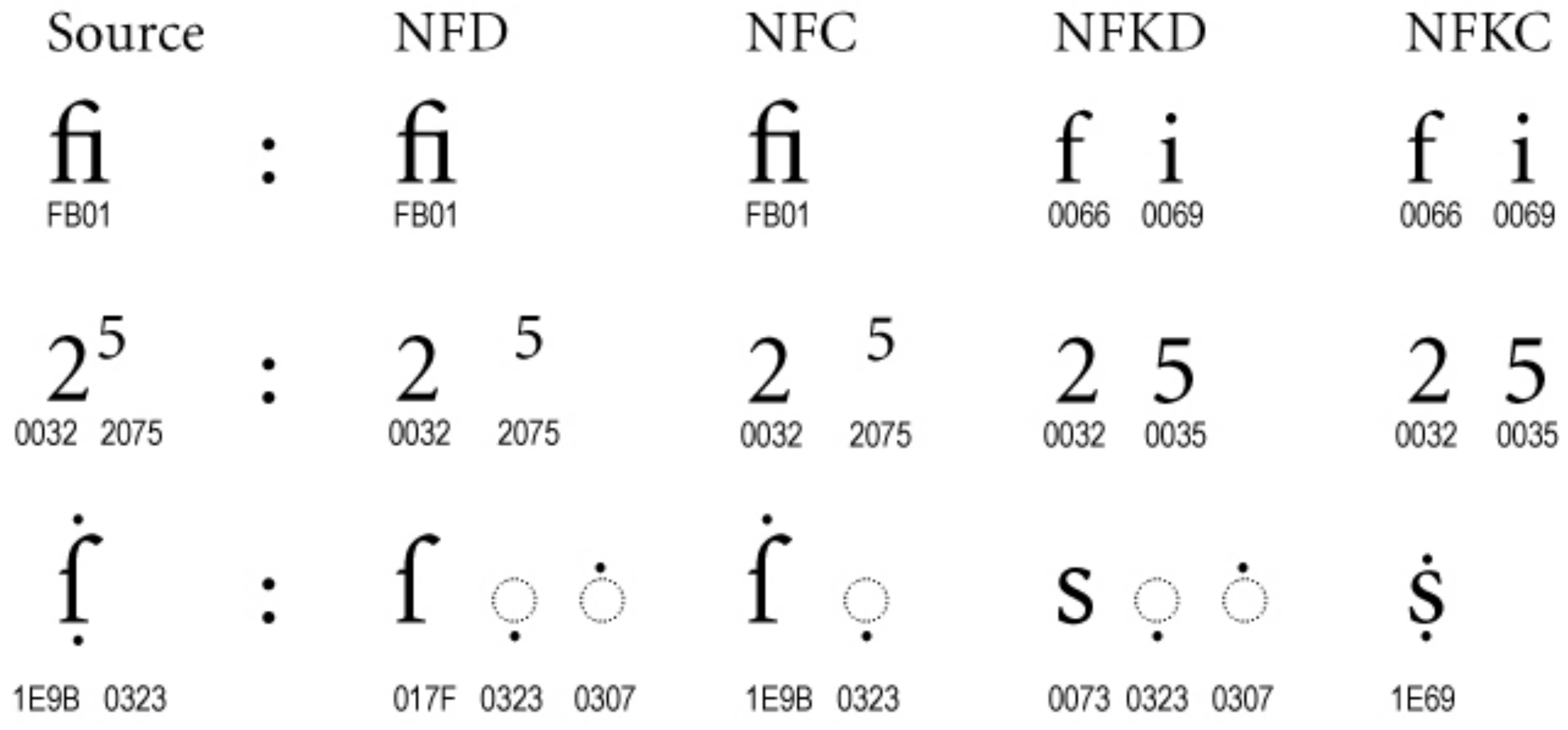
UTF-16

```
H    e    l    l    o    ☺

0048 0065 006c 006c 006f 263a
```

Unicode normalization

NFD (Normalization Form Canonical Decomposition)
NFC (Normalization Form Canonical Composition)
NFKD (Normalization Form Compatibility Decomposition)
NFKC (Normalization Form Compatibility Composition)



Source		NFD	NFC	NFKD	NFKC
ﬁ FB01	:	ﬁ FB01	ﬁ FB01	f i 0066 0069	f i 0066 0069
2^{5} 0032 2075	:	$2\ ^{5}$ 0032 2075	$2\ ^{5}$ 0032 2075	2 5 0032 0035	2 5 0032 0035
ẛ̣ 1E9B 0323	:	ſ ◌̣ ◌̇ 017F 0323 0307	ẛ ◌̣ 1E9B 0323	s ◌̣ ◌̇ 0073 0323 0307	ṩ 1E69

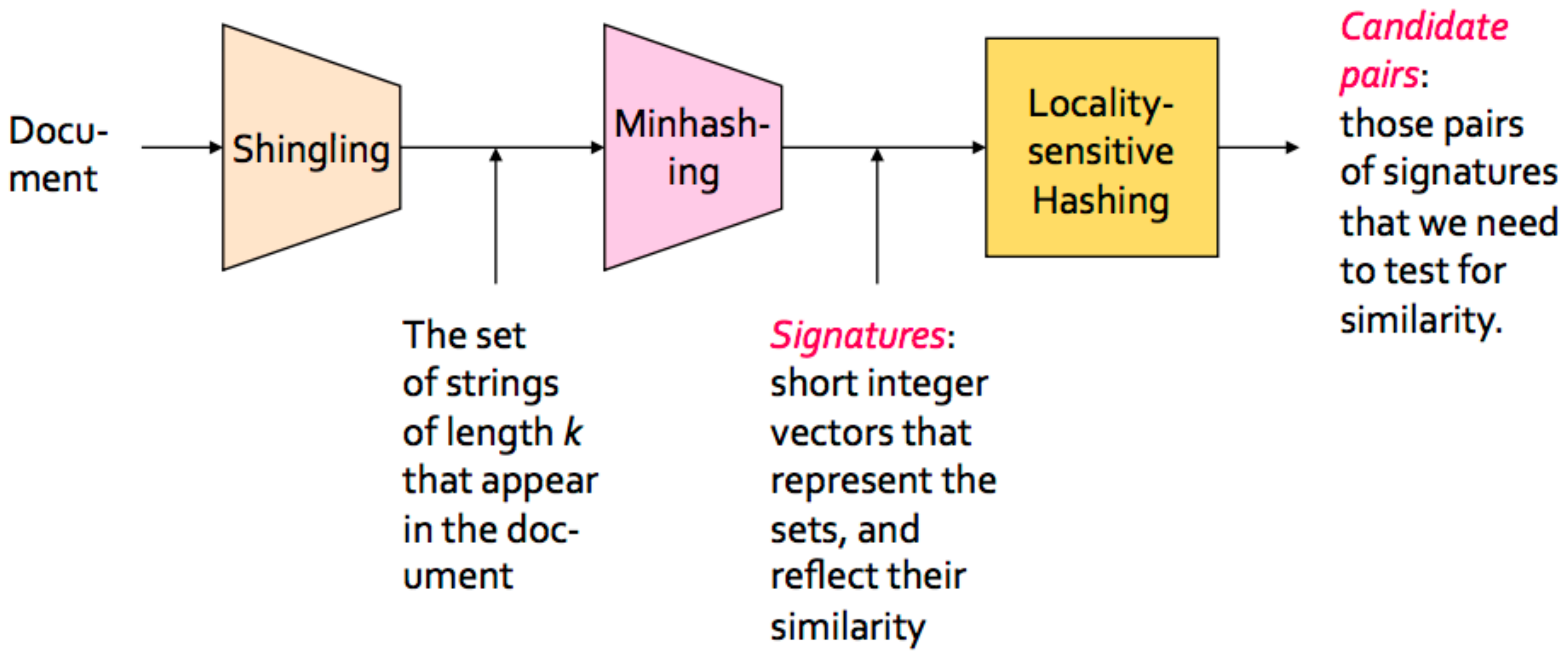
Source: https://unicode.org/

Unicode normalization in Python 3

```
>>> aa = b'\xc4\x81'.decode('utf8')
>>> bb = b'a\xcc\x84'.decode('utf8')
>>> aa
'ā'
>>> bb
'ā'
>>> aa == bb
False
>>> import unicodedata as ud
>>> aa == ud.normalize('NFC',bb)
True
```

Near deduplication

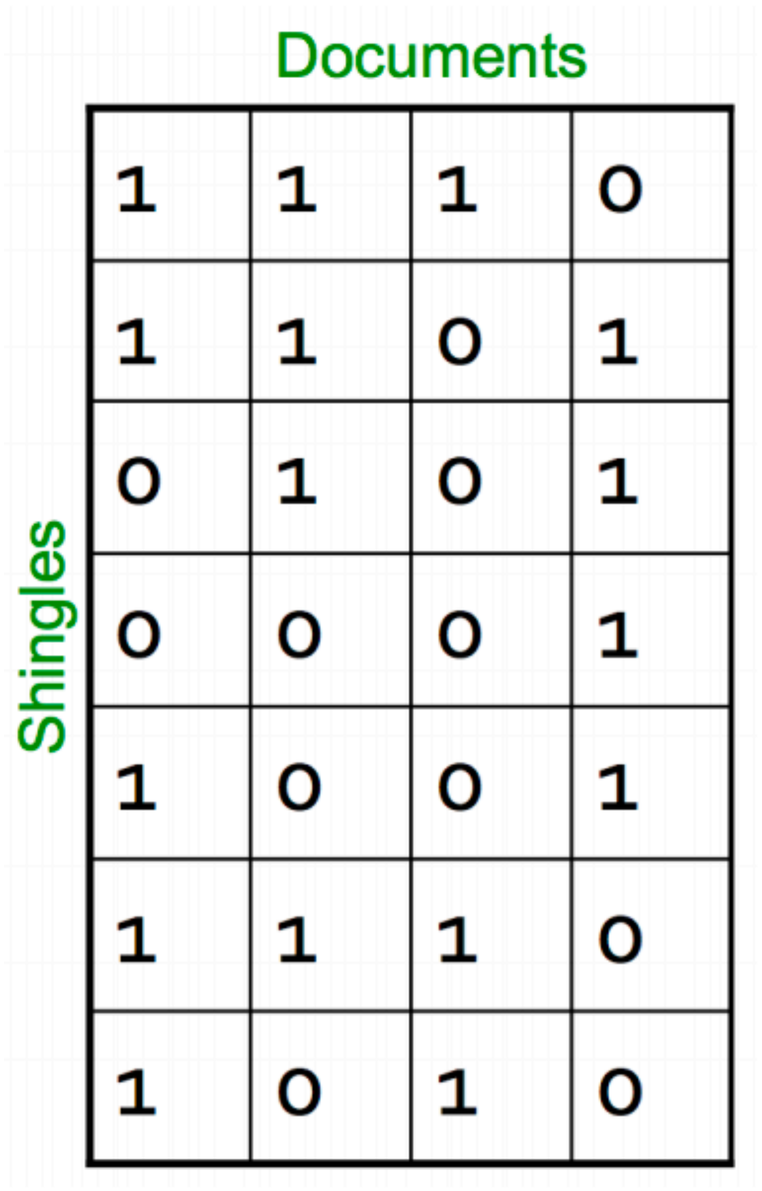
Locality-sensitive hashing



Docu-
ment
Shingling
Minhash-
ing
Locality-
sensitive
Hashing
Candidate pairs: those pairs of signatures that we need to test for similarity.
The set of strings of length *k* that appear in the doc-ument
Signatures: short integer vectors that represent the sets, and reflect their similarity

Source: https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134

Set of shingles (n-grams) as document representation

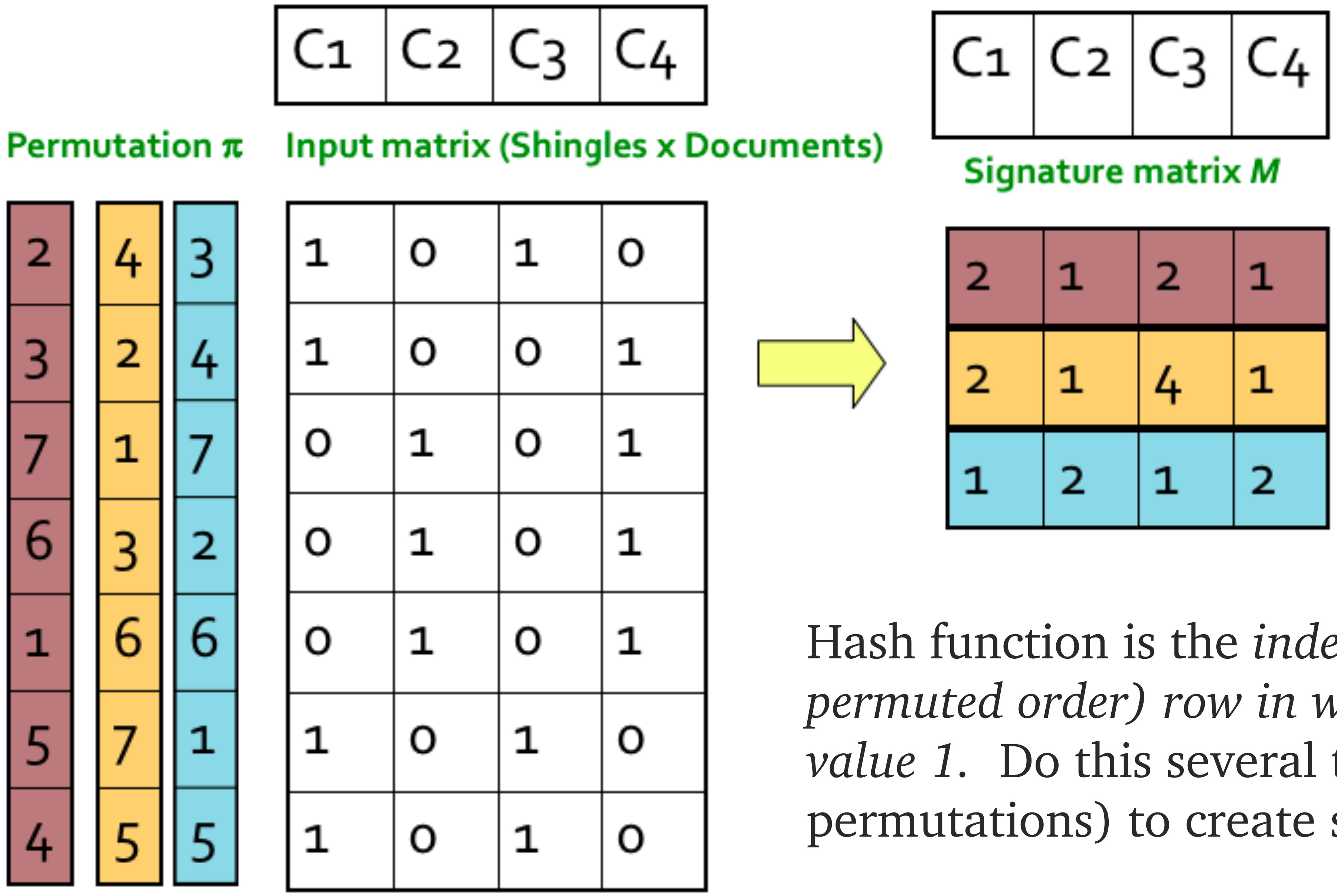

Documents
Shingles

1	1	1	0
1	1	0	1
0	1	0	1
0	0	0	1
1	0	0	1
1	1	1	0
1	0	1	0

Source: https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134

Near deduplication

MinHashing signatures



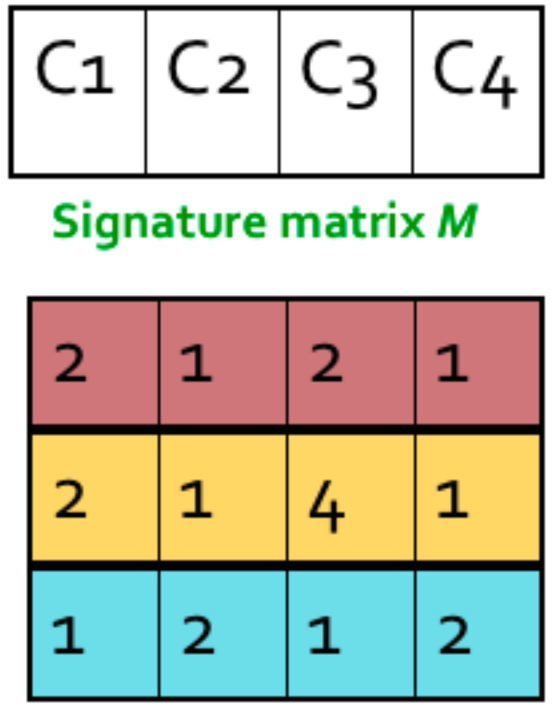
Permutation π			C1	C2	C3	C4
2	4	3	1	0	1	0
3	2	4	1	0	0	1
7	1	7	0	1	0	1
6	3	2	0	1	0	1
1	6	6	0	1	0	1
5	7	1	1	0	1	0
4	5	5	1	0	1	0

C1	C2	C3	C4
2	1	2	1
2	1	4	1
1	2	1	2

Hash function is the *index of the first (in the permuted order) row in which column C has value 1*. Do this several time (use different permutations) to create signature of a column.

Source: https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134

Jaccard similarity and MinHashing signatures



C1	C2	C3	C4

Signature matrix *M*

2	1	2	1
2	1	4	1
1	2	1	2

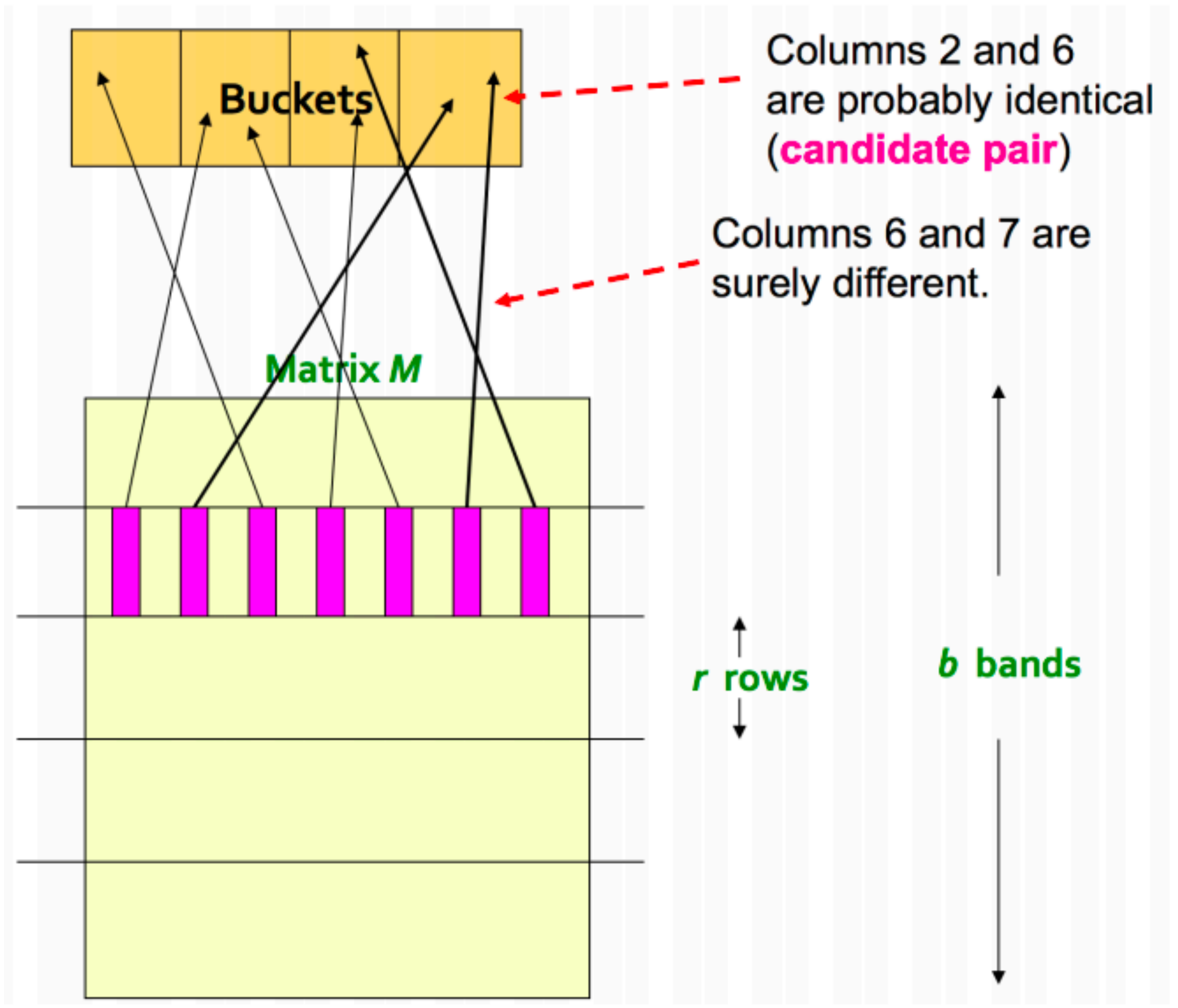
The similarity of the signatures is the fraction of the min-hash functions (rows) in which they agree. So the similarity of signature for C1 and C3 is 2/3 as 1st and 3rd row are same.

Claim: $P[h_\pi(C_1) = h_\pi(C_2)] = \mathrm{sim}(C_1, C_2)$

Source: https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134

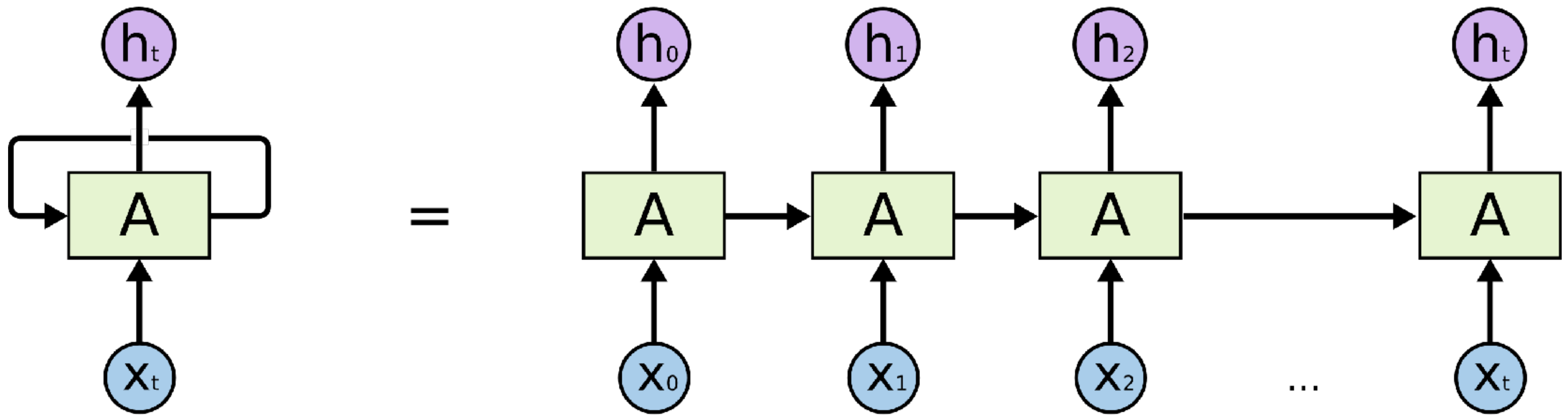
Near deduplication

Locality-sensitive hashing


Buckets
Columns 2 and 6 are probably identical (candidate pair)
Columns 6 and 7 are surely different.
Matrix *M*
r rows
b bands

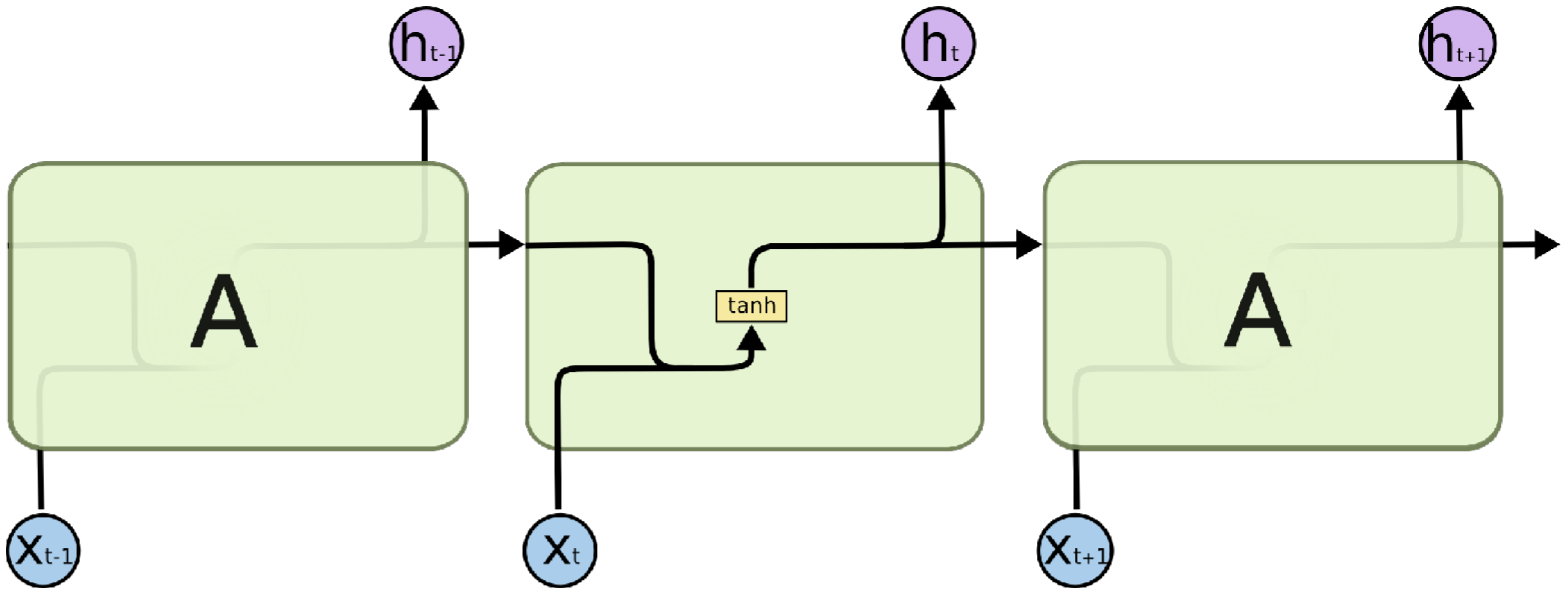
Source: https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134

Recurrent Neural networks 1/2



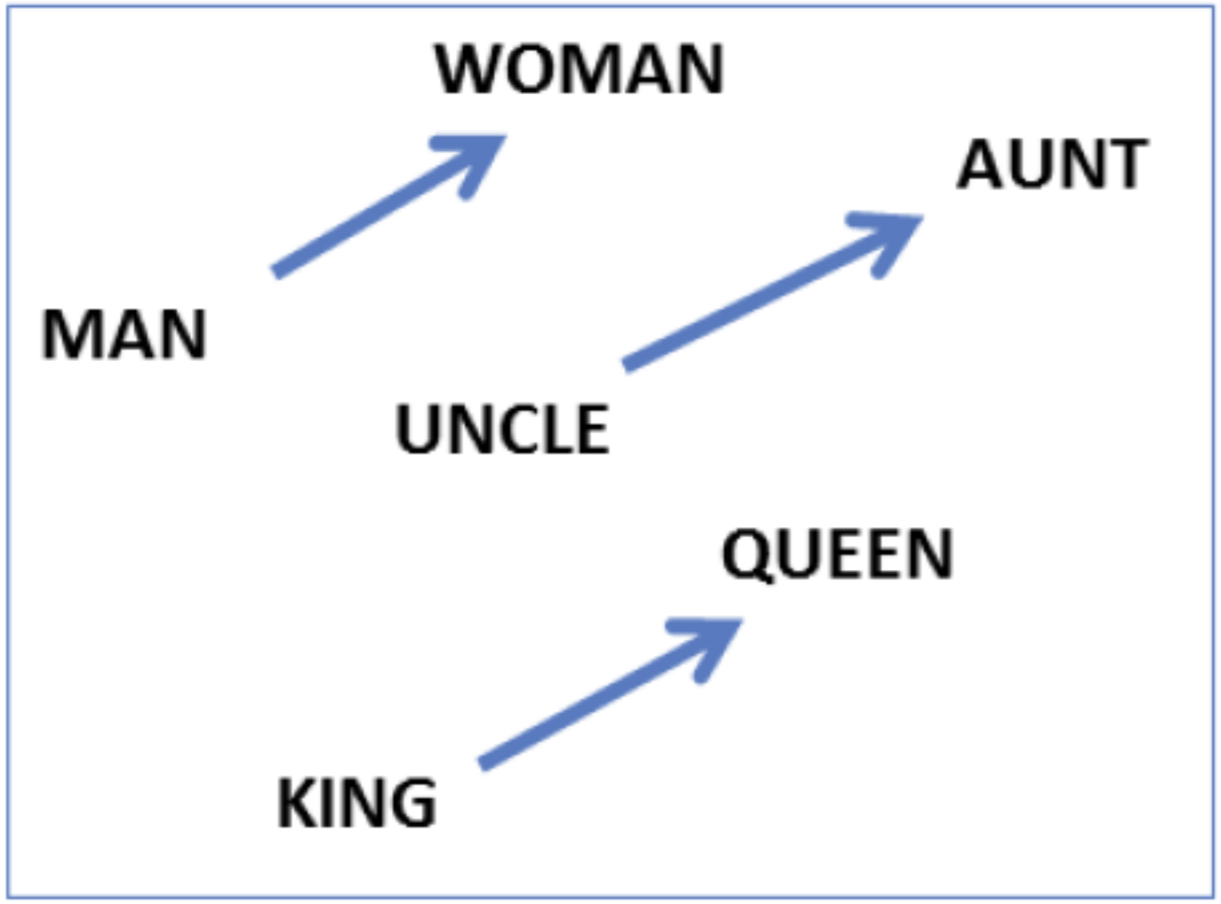
source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

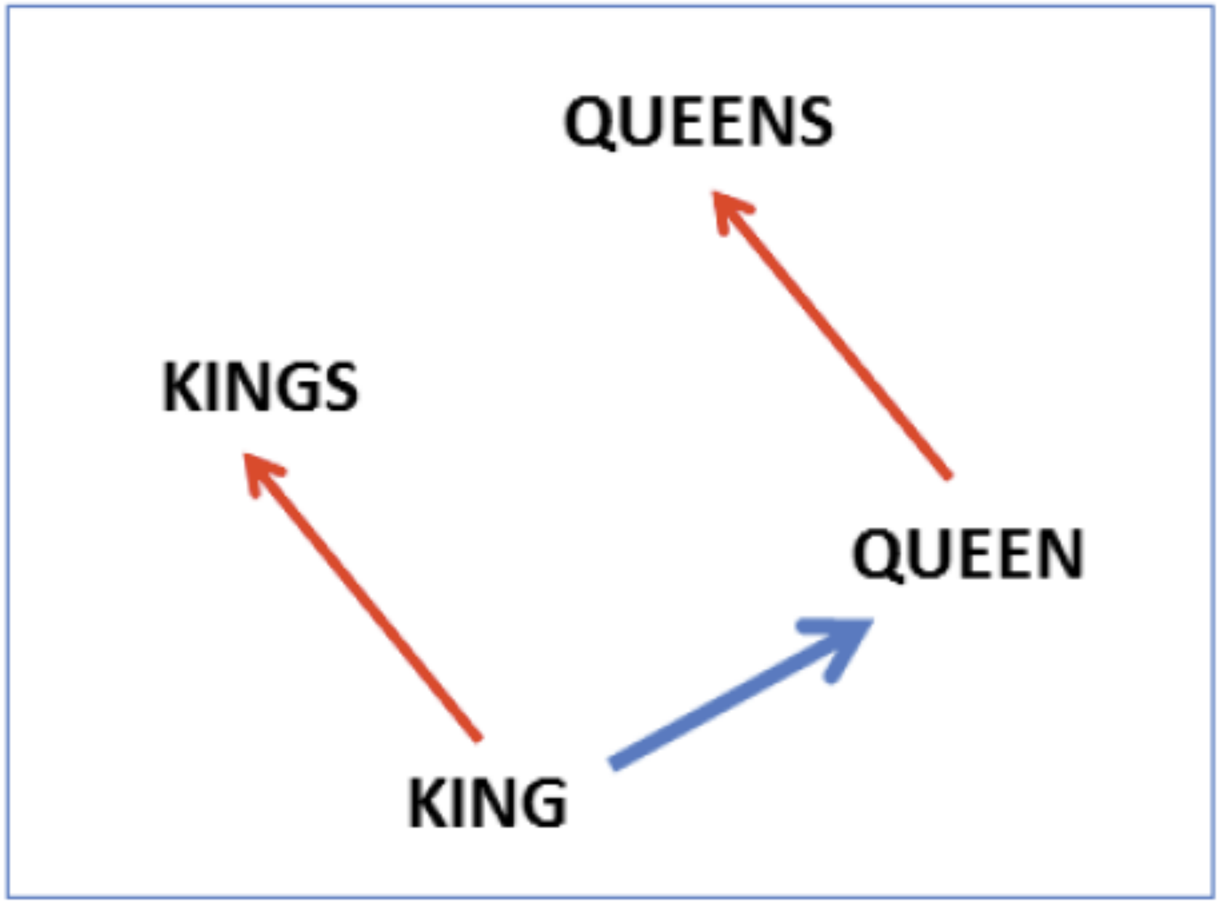
Recurrent Neural Networks 2/2



https://www.tensorflow.org

word2vec

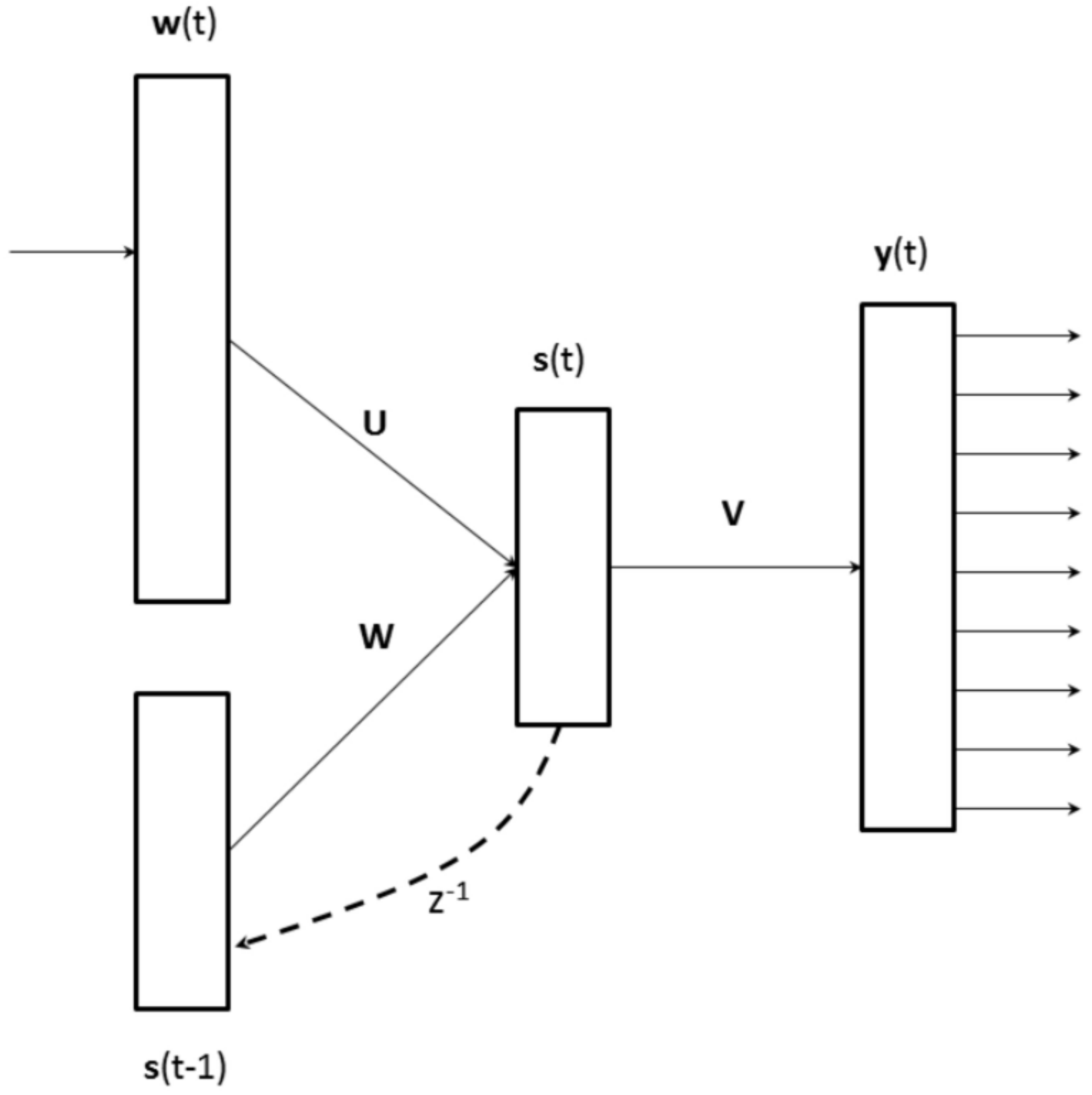




king is to **kings** as **queen** to ***?***.

v(**kings**) - v(**king**) = v(**queens**) - v (**queen**)

Recurrent Neural Network Language Modeling Toolkit



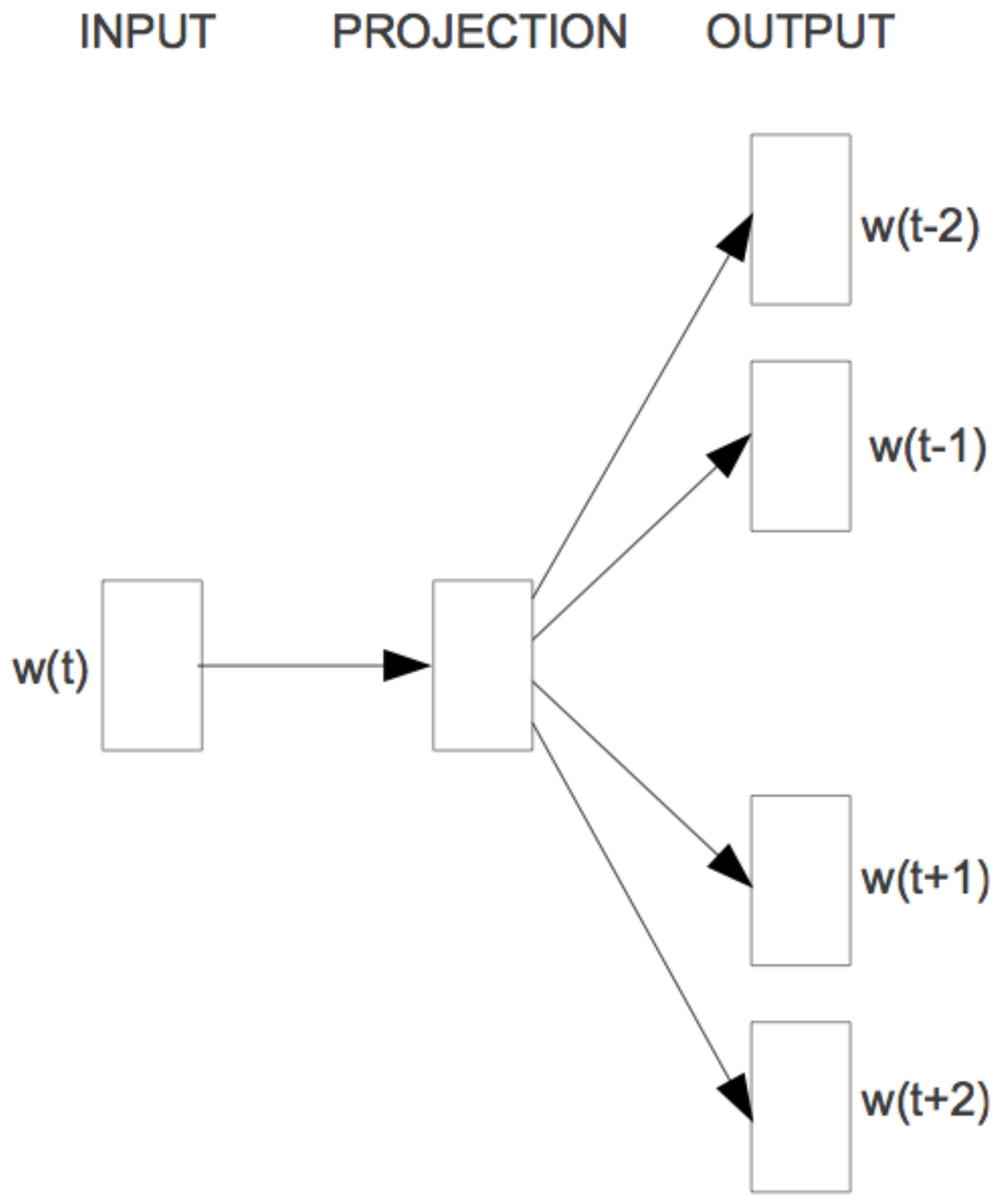
$$\mathbf{s}(t) = f\left(\mathbf{U}\mathbf{w}(t) + \mathbf{W}\mathbf{s}(t-1)\right)$$

$$\mathbf{y}(t) = g\left(\mathbf{V}\mathbf{s}(t)\right),$$

$$f(z) = \frac{1}{1+e^{-z}}, \quad g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}.$$

source: http://www.fit.vutbr.cz/~imikolov/rnnlm/

The Skip-gram model



Skip-gram improvements

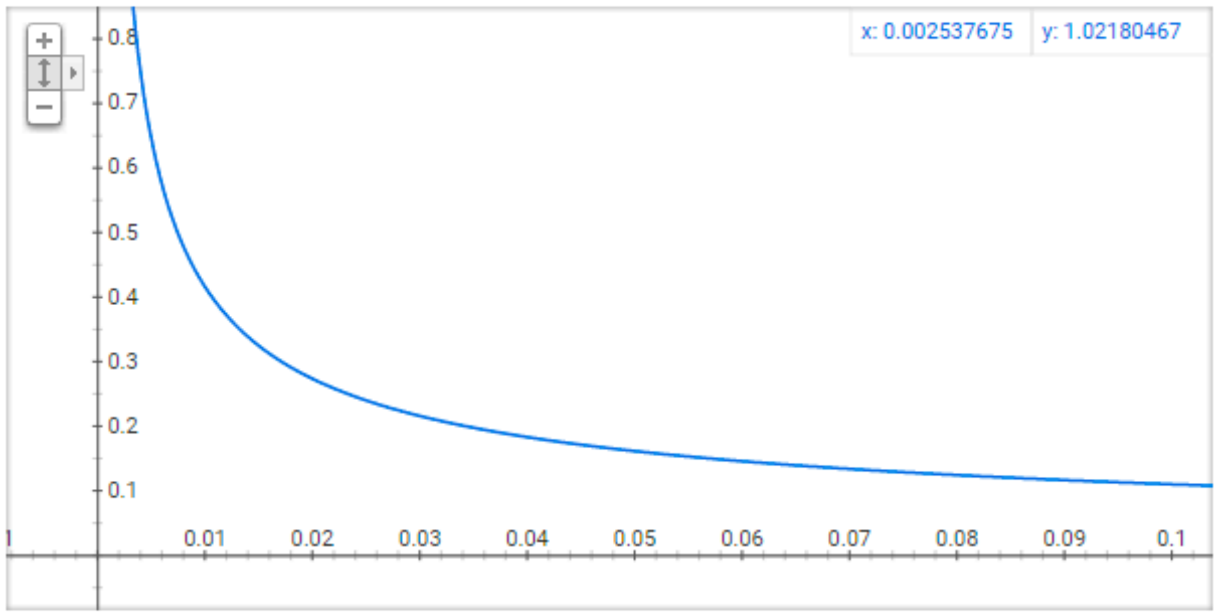
Subsampling frequent inputs

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

z(w) Relative frequency of word *w*

P(w) Probability of keeping word *w*

Graph for (sqrt(x/0.001)+1)*0.001/x



Negative sampling

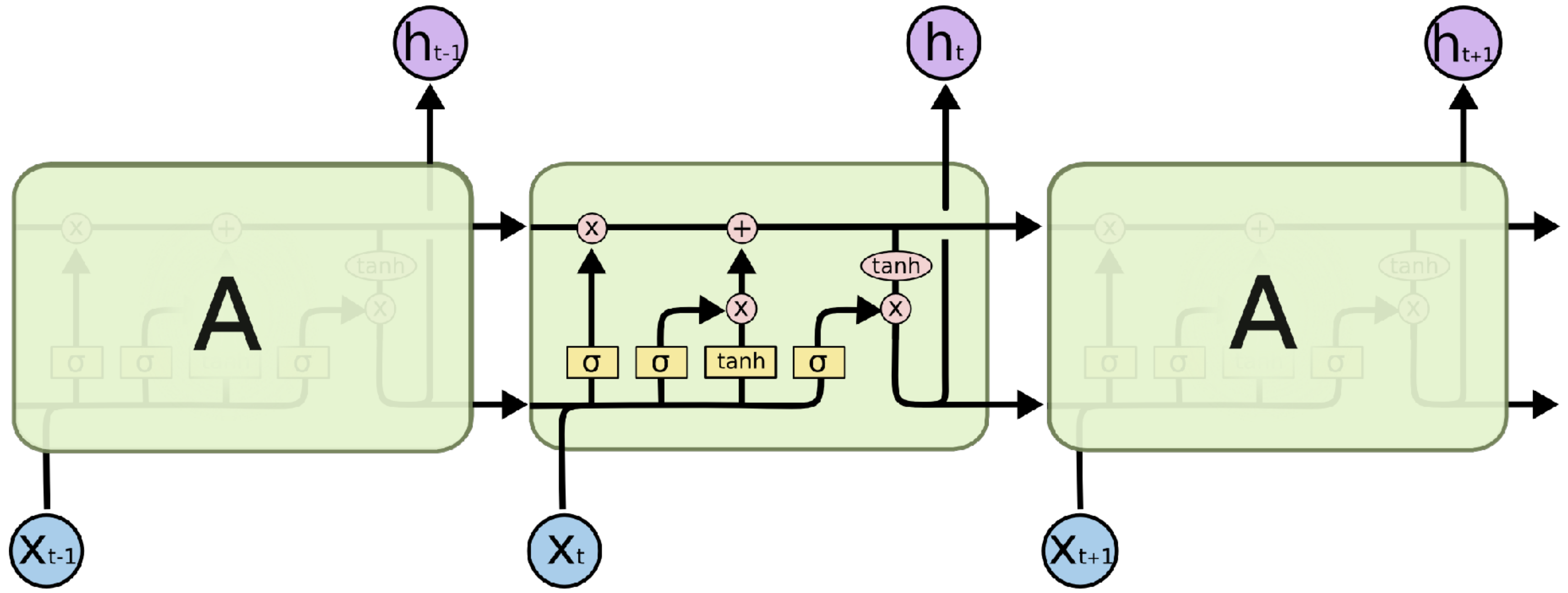
We select only 5-20 negative samples in the loss function.
The probability of picking a word *w* is given by *z(w).*

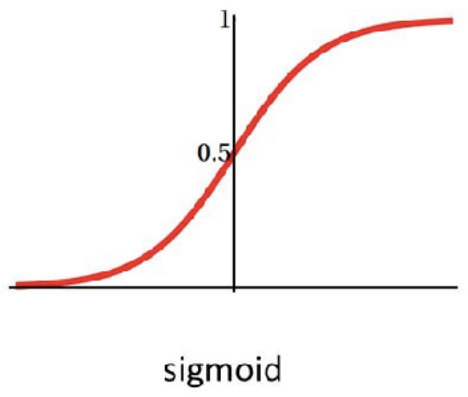
http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

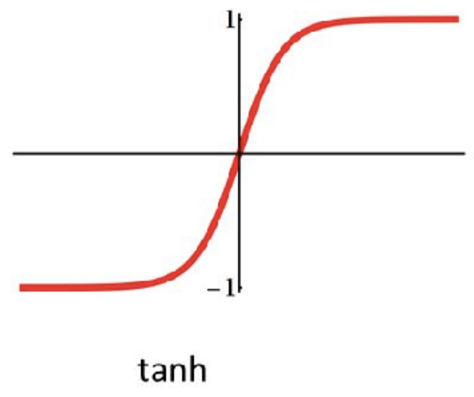
Experiments with word2vec

01-Review-classification-w2v-assignment.ipynb

Long Short-Term Memory

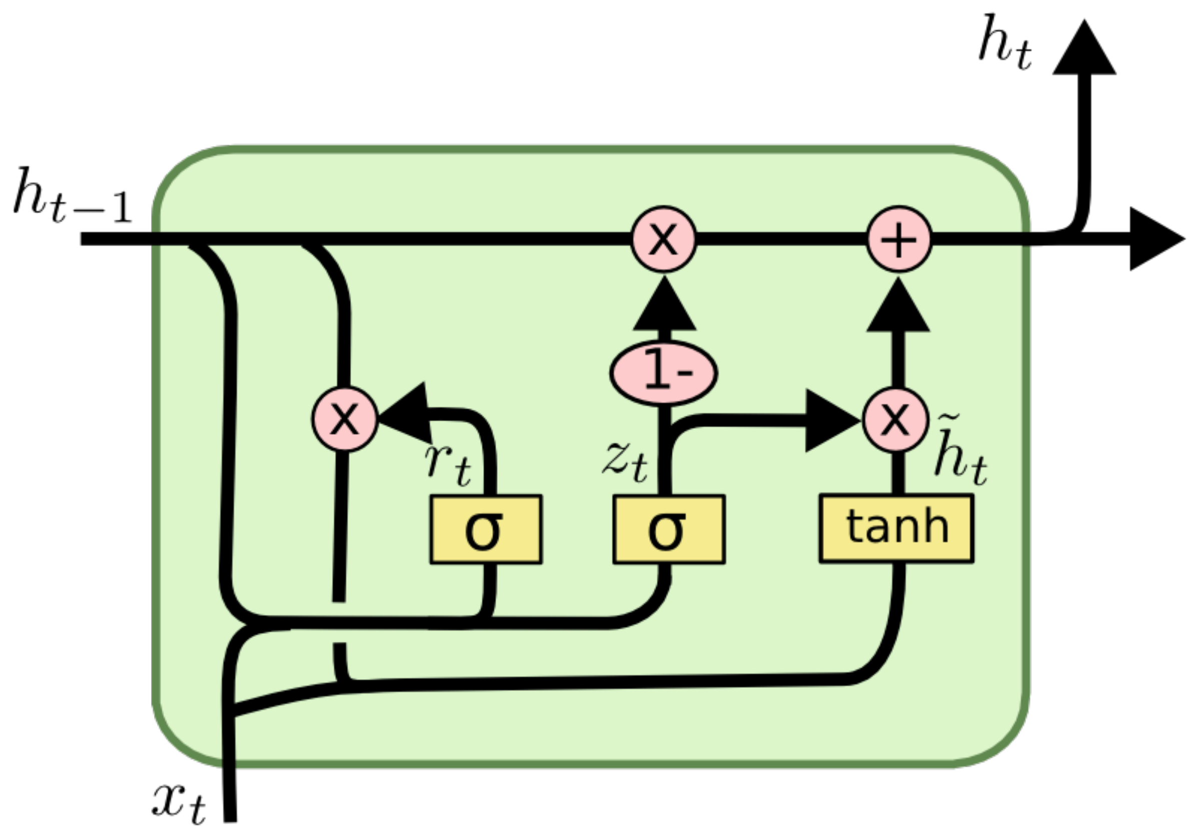






Zdroj: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Unit



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

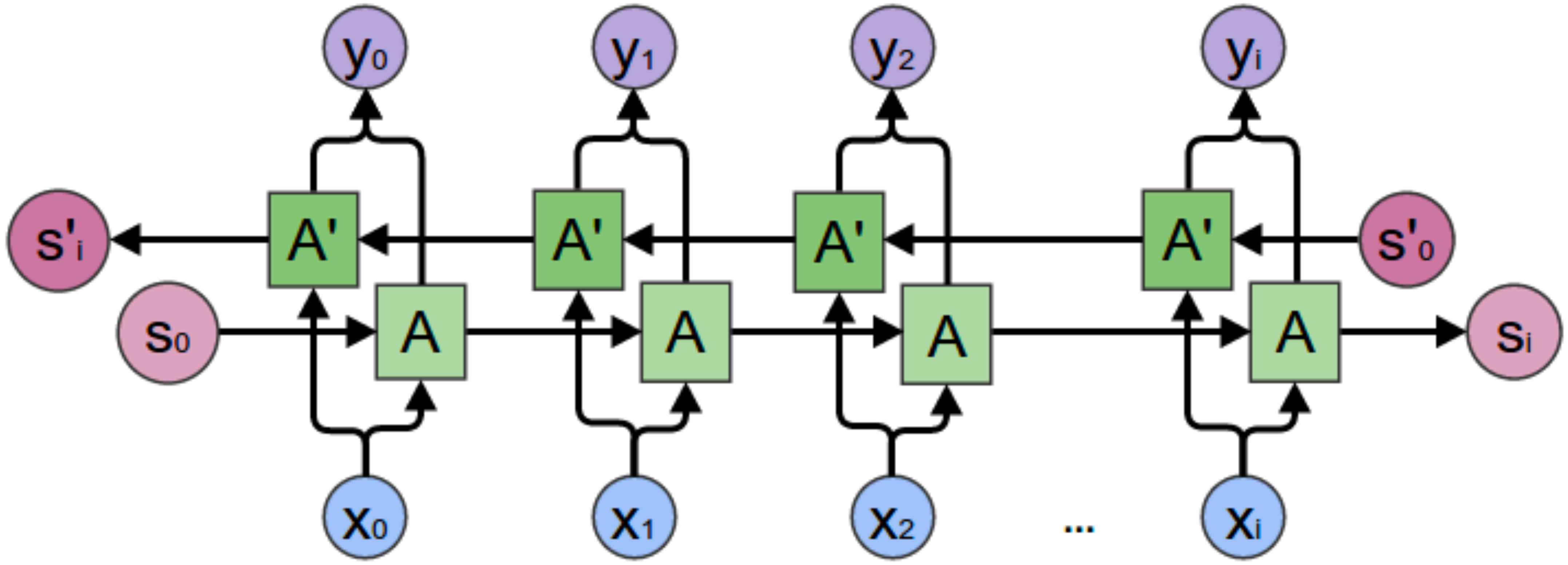
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

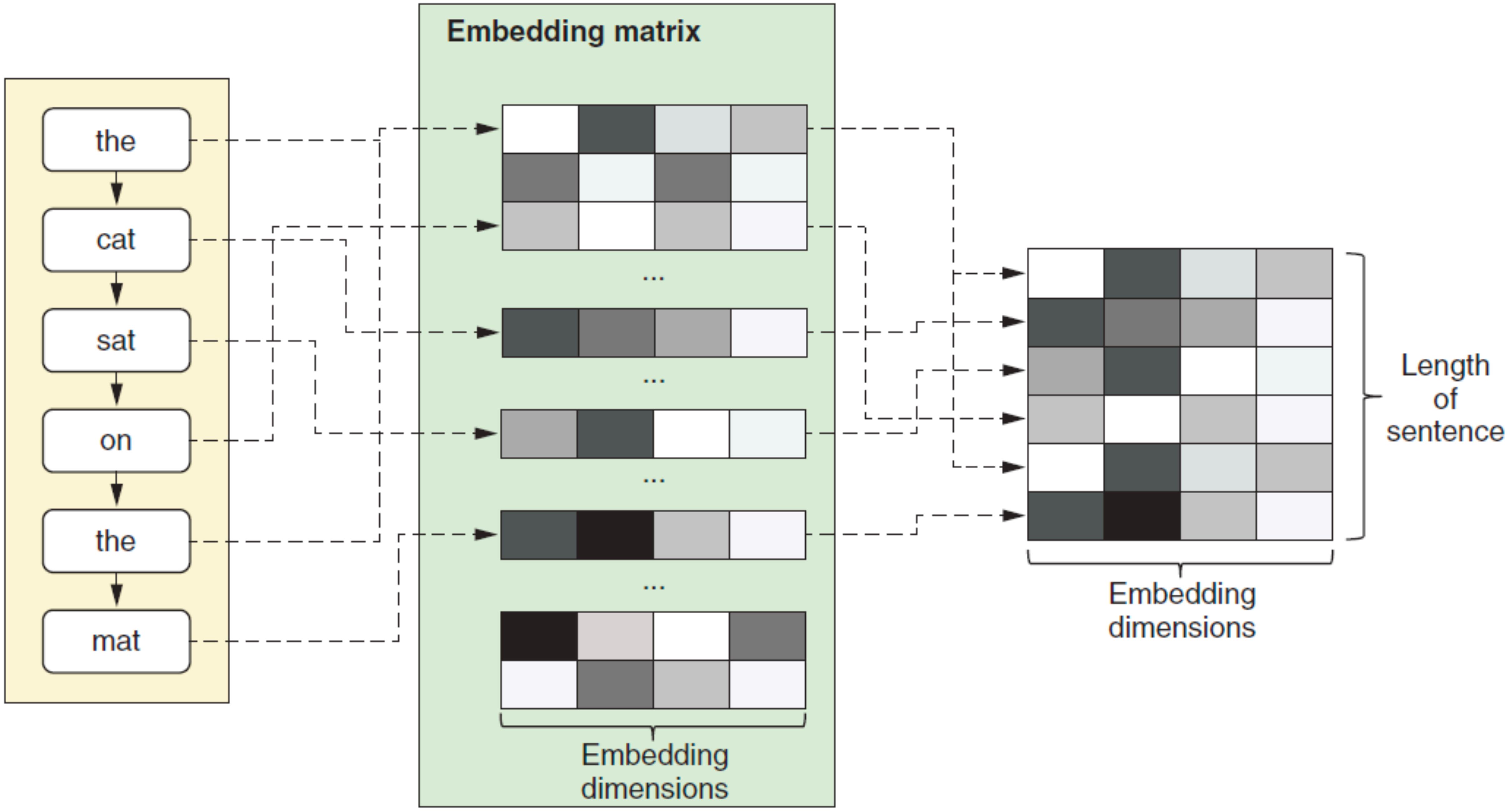
Zdroj: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Bidirectional recursive layer in Keras



http://colah.github.io/posts/2015-09-NN-Types-FP/

Embedding layer in Keras



https://medium.com/@djajafer/multi-class-text-classification-with-keras-and-lstm-4c5525bef592

Text classification with bidirectional LSTM

02-Review-classification-LSTM.ipynb

Traditional tokenization

NLTK tokenizers

```
>>> from nltk.tokenize import word_tokenize #simple
>>> from nltk.tokenize.moses import MosesTokenizer #enables detokenization
>>> from nltk.tokenize import ToktokTokenizer #fast
>>>
>>> moses = MosesTokenizer()
>>> toktok = ToktokTokenizer()
>>>
>>> text = "Welcome to Machine Learning College."
>>> print(word_tokenize(text))
>>> print (moses.tokenize(text))
>>> print (toktok.tokenize(text))
['Welcome', 'to', 'Machine', 'Learning', 'College', '.']
['Welcome', 'to', 'Machine', 'Learning', 'College', '.']
['Welcome', 'to', 'Machine', 'Learning', 'College', '.']
```

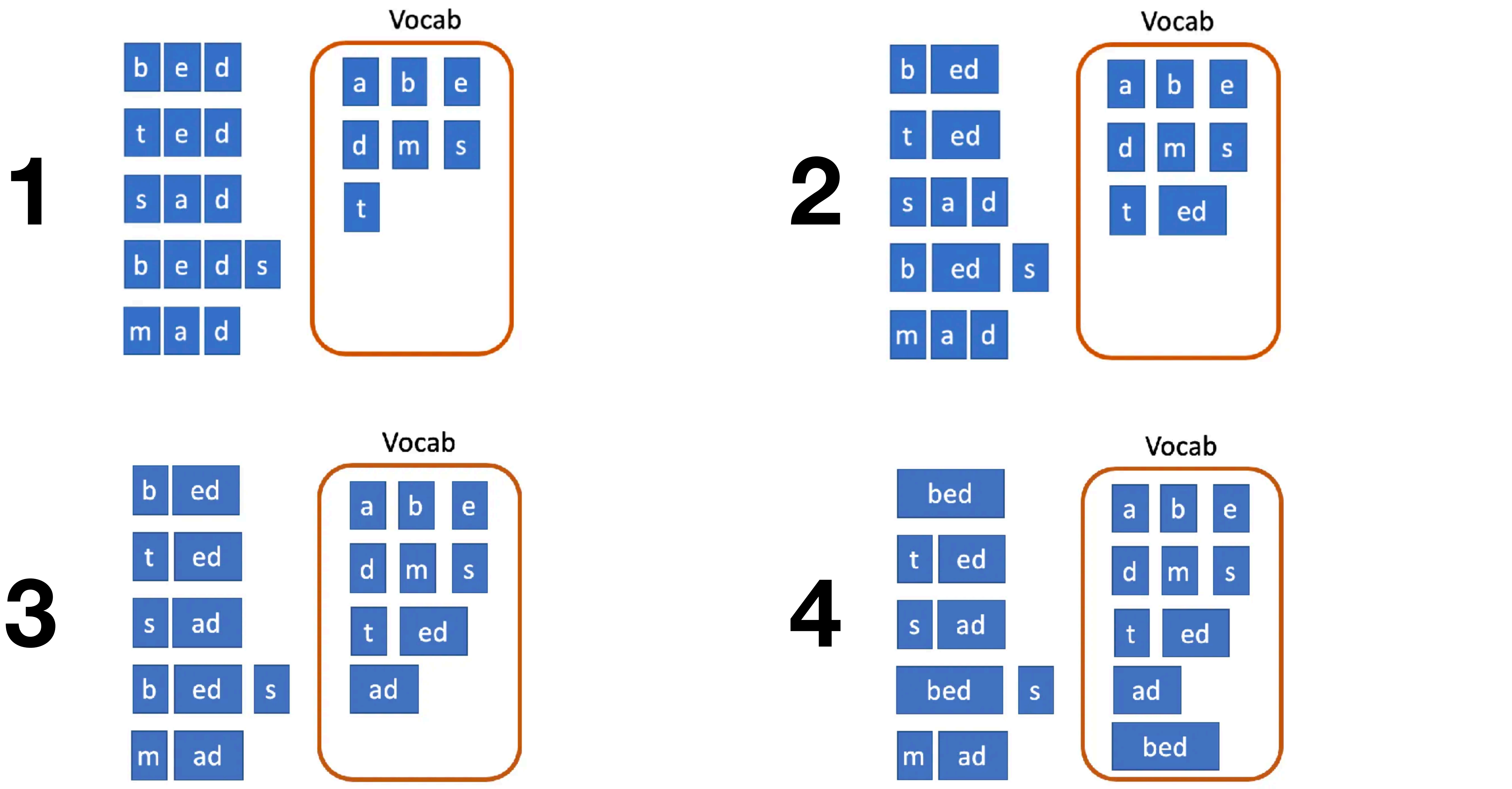
Traditional tokenization

SpaCy tokenizer

```
>>> import spacy
>>> sp = spacy.load('en_core_web_sm')
>>> tokens = sp("Welcome to Machine Learning College.")
>>>
>>> [word.text for word in tokens]
['Welcome', 'to', 'Machine', 'Learning', 'College', '.']
```

Subword tokenization

Byte-pair encoding



Source: https://mlexplained.com/2019/11/06/a-deep-dive-into-the-wonderful-world-of-preprocessing-in-nlp/

Subword tokenization

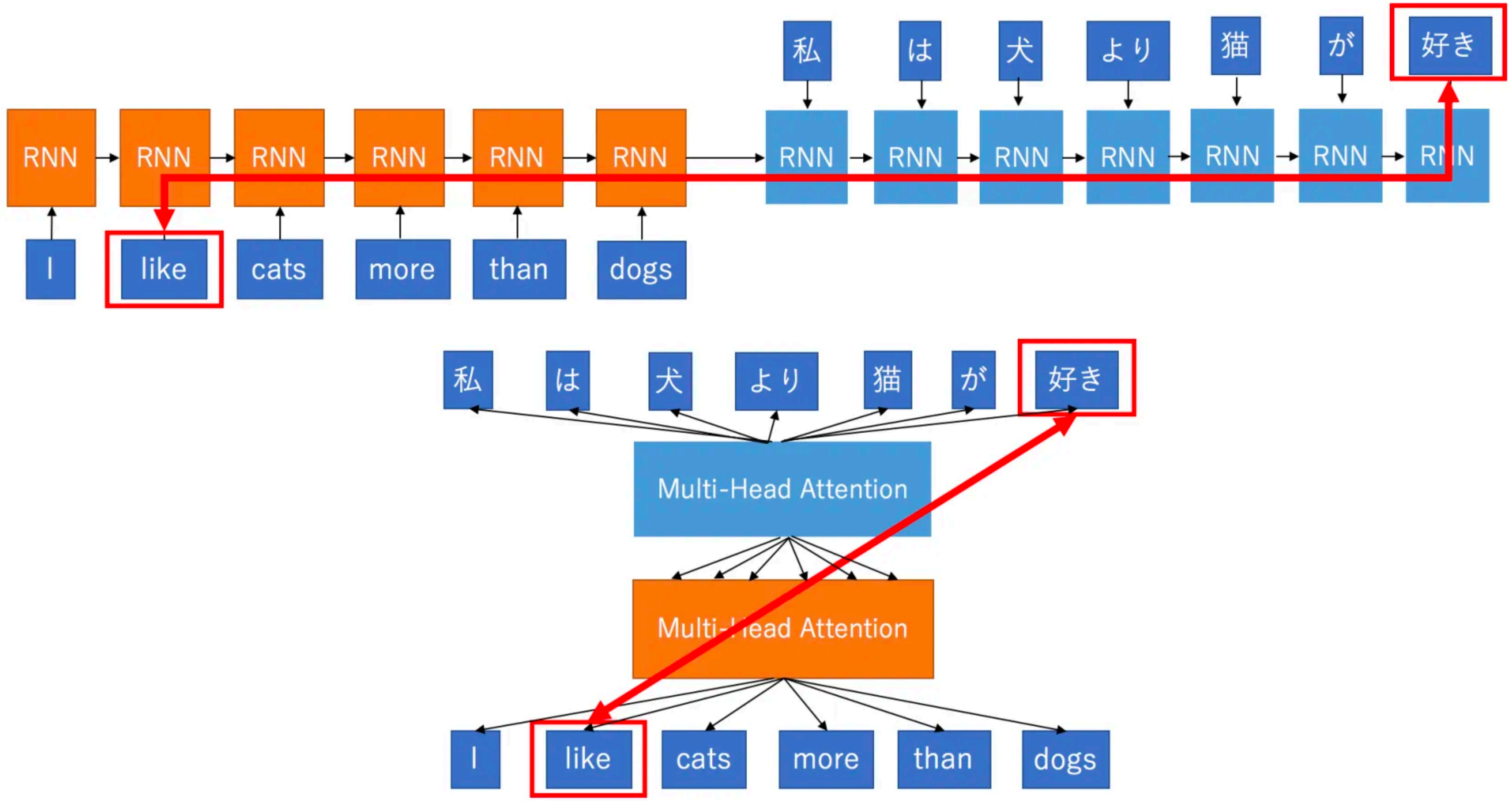
Wordpiece and sentencepiece tokenization

Merges bigrams with maximum mutual information instead of maximum frequency.

$$I(x,y) = \log\left(\frac{p(x,y)}{p(x)\,p(y)}\right)$$

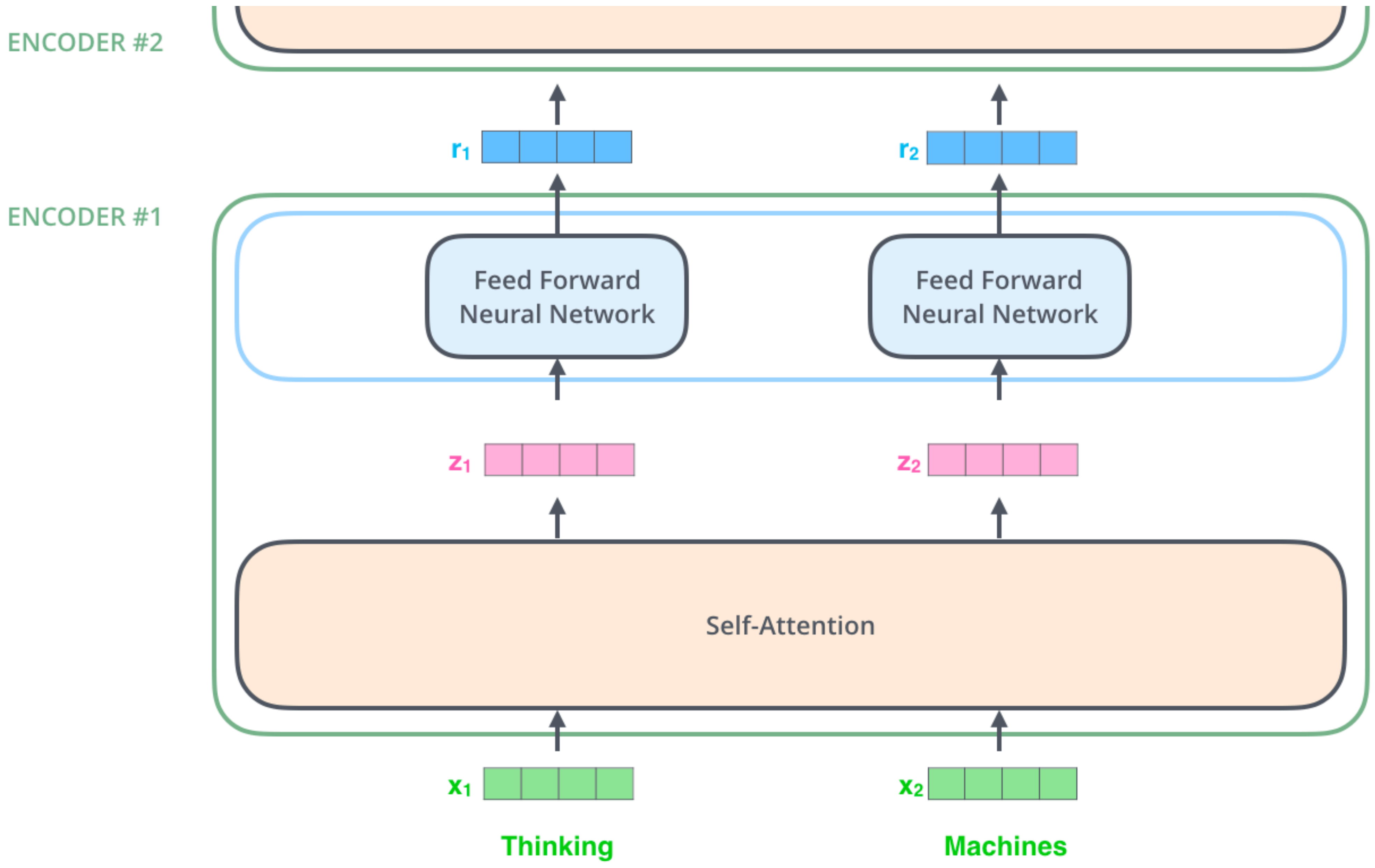
playing -> play, ##ing

RNN vs. Transformer



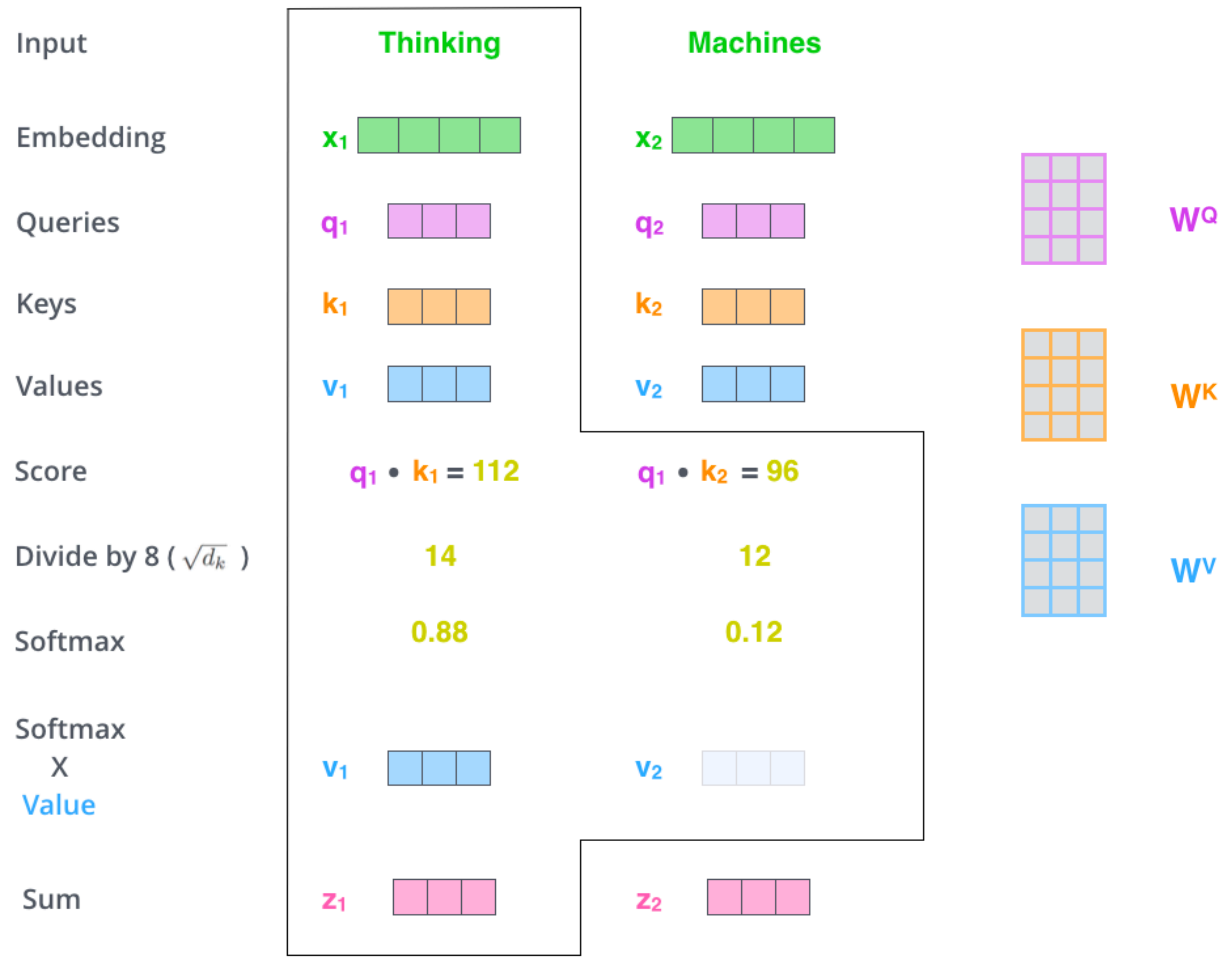
source: www.mlexplained.com

Attention is all you need



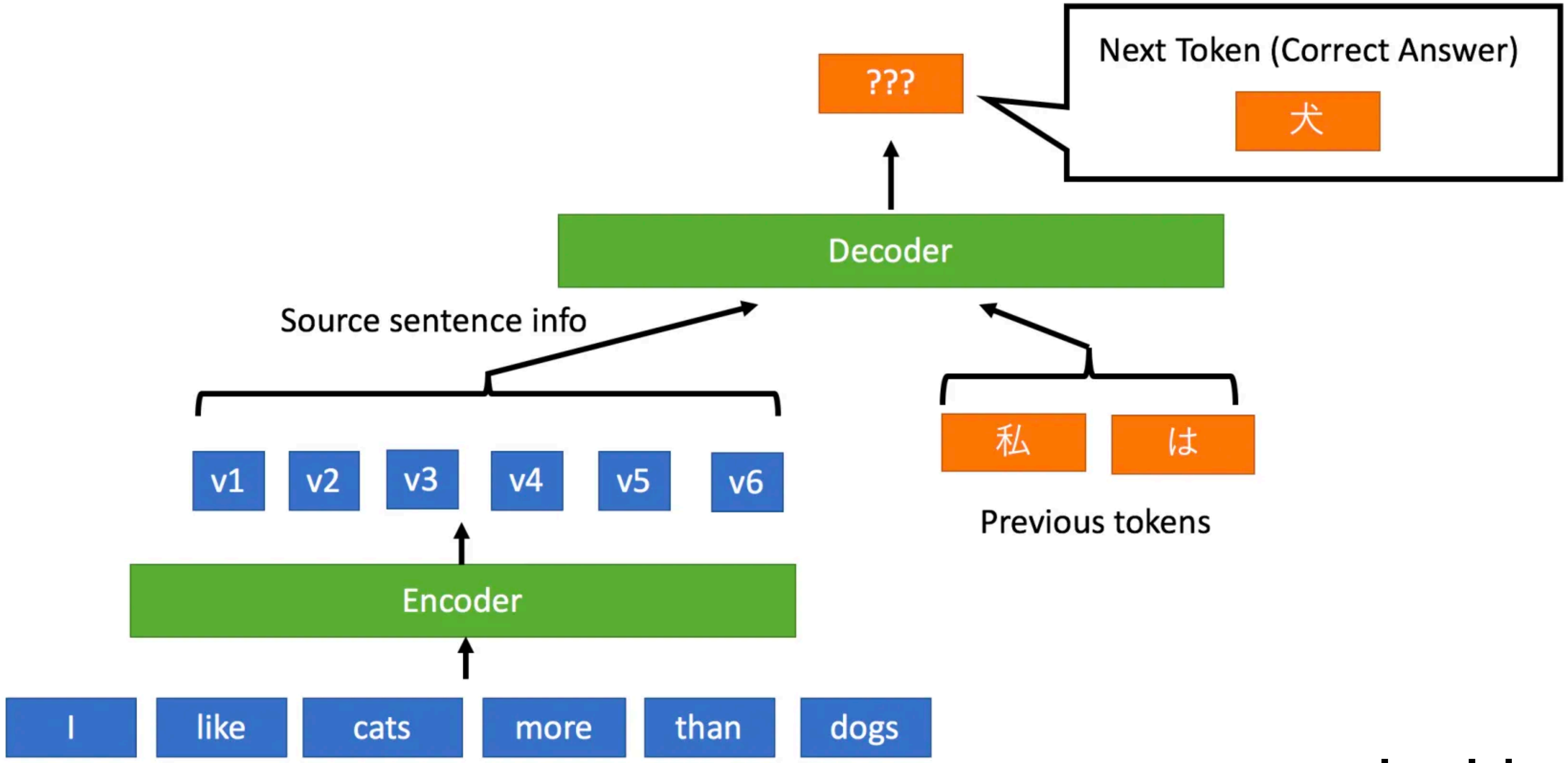
http://jalammar.github.io/illustrated-transformer/

Self-attention



http://jalammar.github.io/illustrated-transformer/

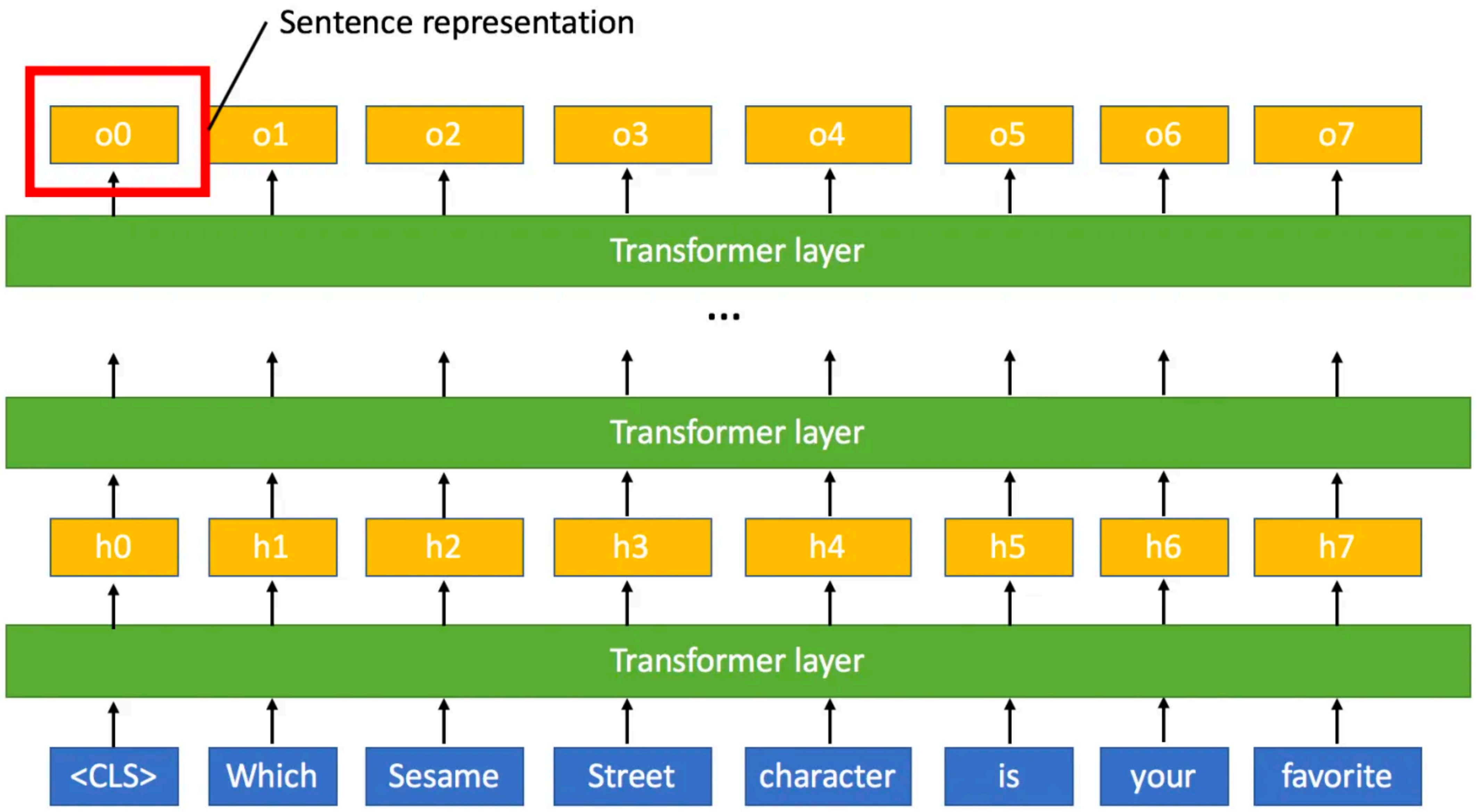
Translation with Transformers



GPT-3 Language model

https://beta.openai.com/

BERT (classification)



Text classification using BERT

03-Review-classification-BERT.ipynb