Machine learning approaches for Natural language processing

Toxic Comment Classification Challenge

Sergii Makarevych
sermakarevich@gmail.com

Toxic Comment Classification Challenge

Discussing things you care about can be difficult. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.

The Conversation AI team, a research initiative founded by Jigsaw and Google (both a part of Alphabet) are working on tools to help improve online conversation. One area of focus is the study of negative online behaviors, like toxic comments (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion). So far they've built a range of publicly available models served through the Perspective API, including toxicity. But the current models still make errors, and they don't allow users to select which types of toxicity they're interested in finding (e.g. some platforms may be fine with profanity, but not with other types of toxic content).

In this competition, you're challenged to build a multi-headed model that's capable of detecting different types of of toxicity like threats, obscenity, insults, and identity-based hate better than Perspective's current models. You'll be using a dataset of comments from Wikipedia's talk page edits. Improvements to the current model will hopefully help online discussion become more productive and respectful.

Disclaimer: the dataset for this competition contains text that may be considered profane, vulgar, or offensive.

Toxic comments example

One comment might belong to multiple categories.

Toxic

'COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK'

Severe Toxic

Stupid peace of shit stop deleting my stuff asshole go die and fall in a hole go to hell!

Obscene

You are gay or antisemmitian?

Threat

I think that your a Fagget get a oife and burn in Hell I hate you 'm sorry we cant have any more sex i'm running out of conndoms

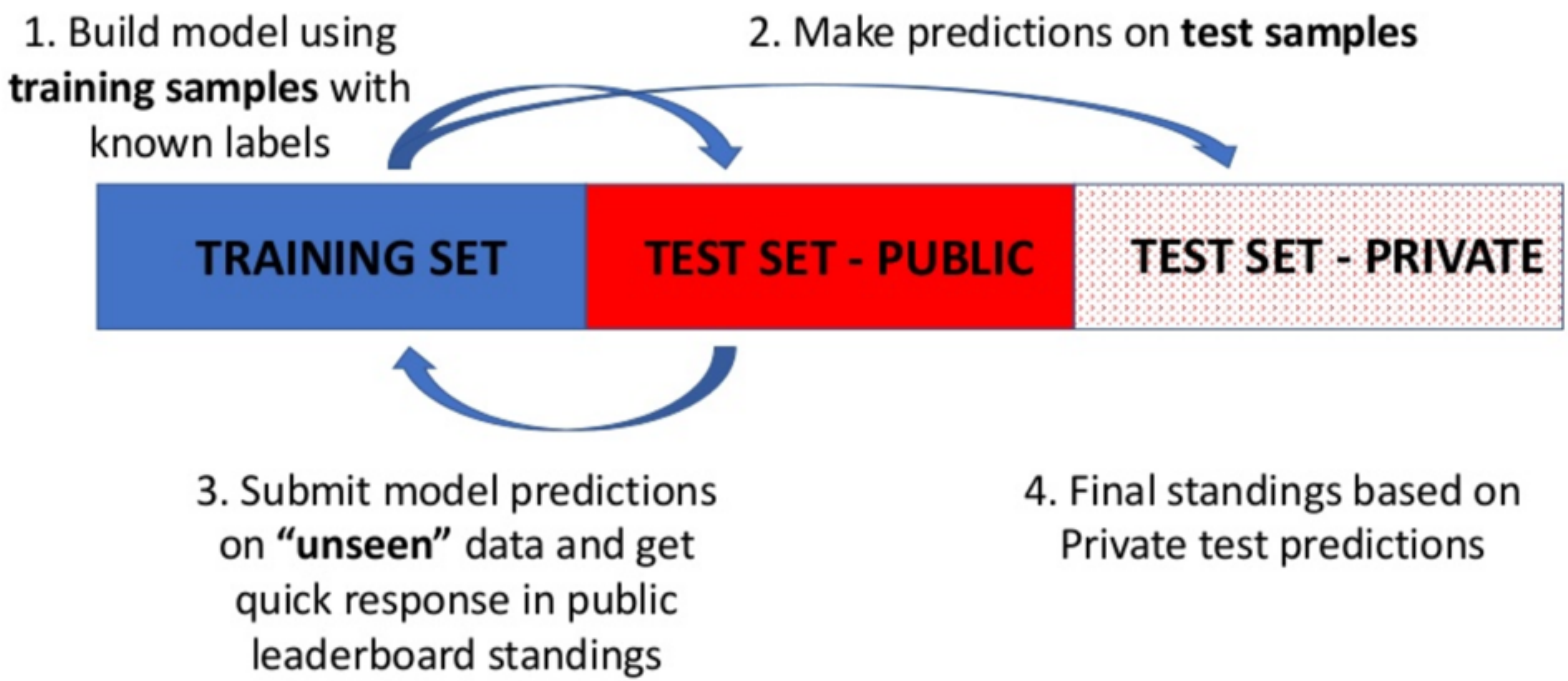
Insult

FUCK YOUR FILTHY MOTHER IN THE ASS, DRY!

Identity hate

Kill all niggers. I have hard, that others have said this.. should this be included? That racists sometimes say these.

HOW DOES IT WORK



from radar`s presentation

Machine learning approach

- Clear data
  - lemmatization
  - contractions
  - tokenization
- Transform data
  - counts
  - TFIDF
  - n-grams
  - NB features
- Apply model
  - Logistic regression
- Words polarity based on LR weights

Lemmatization

Lemmatization usually aiming to remove inflectional endings and to return the base or dictionary form of a word, which is known as the lemma, with the use of a vocabulary and morphological analysis of words.

```
>>> print(wnl.lemmatize('dogs'))
dog
>>> print(wnl.lemmatize('churches'))
church
>>> print(wnl.lemmatize('aardwolves'))
aardwolf
>>> print(wnl.lemmatize('abaci'))
abacus
>>> print(wnl.lemmatize('hardrock'))
hardrock
```

```
MORPHOLOGICAL_SUBSTITUTIONS = {
    NOUN: [('s', ''), ('ses', 's'), ('ves', 'f'), ('xes', 'x'),
           ('zes', 'z'), ('ches', 'ch'), ('shes', 'sh'),
           ('men', 'man'), ('ies', 'y')],
    VERB: [('s', ''), ('ies', 'y'), ('es', 'e'), ('es', ''),
           ('ed', 'e'), ('ed', ''), ('ing', 'e'), ('ing', '')],
    ADJ: [('er', ''), ('est', ''), ('er', 'e'), ('est', 'e')],
    ADV: []}
```

https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

Contractions

```
contractions = {
    "ain't": "am not",
    "aren't": "are not",
    "can't": "cannot",
    "can't've": "cannot have",
    "'cause": "because",
    "could've": "could have",
    "couldn't": "could not",
    "couldn't've": "could not have",
    "didn't": "did not",
    "doesn't": "does not",
    "don't": "do not",
    "hadn't": "had not",
    "hadn't've": "had not have",
    "hasn't": "has not",
    "haven't": "have not",
    "he'd": "he had",
    "he'd've": "he would have",
    "he'll": "he shall / he will",
    "he'll've": "he shall have / he will have",
    "he's": "he has",
    "how'd": "how did",
    "how'd'y": "how do you",
    "how'll": "how will",
    "how's": "how has",
    "I'd": "I had",
    "I'd've": "I would have",
    "I'll": "I will",
    "I'll've": "I will have",
    "I'm": "I am",
    "I've": "I have",
    "isn't": "is not",
```

Tokenization

Splitting text into tokens (symbols/chars)

```python
token_pattern = re.compile('\w{1,}')
def tokenize(s):
    return token_pattern.findall(s)

token_pattern.findall('Hey guys!Whats up_there?')
```

```
['Hey', 'guys', 'Whats', 'up_there']
```

```python
nltk.word_tokenize('Hey guys!Whats up_there?')
```

```
['Hey', 'guys', '!', 'Whats', 'up_there', '?']
```

```python
re_tok = re.compile(f'([{string.punctuation}“”¨«»®´·º½¾¿¡§£₤‘’])')
def tokenize(s):
    return re_tok.sub(r' \1 ', s).split()

tokenize('Hey guys!Whats up_there?')
```

```
['Hey', 'guys', '!', 'Whats', 'up', '_', 'there', '?']
```

Counts

To generate tokens count - number of times a word/ chars occured in each set of the corpus. after trasformation we have a matrix of the same number of rows and number of columns equal to number of unique words/tokens in the corpus unless we decided to truncate it. In this case words with low frequency are out of the analysis.

```
corpus = [
     'This is the first document.',
     'This is the second second document.',
     'And the third one.',
     'Is this the first document?',
]
```

```
model = CountVectorizer()
X = model.fit_transform(corpus)
pd.DataFrame(X.todense(),
             columns=model.vocabulary_)
```

executed in 14ms, finished 15:43:12 2018-03-22

	this	is	the	first	document	second	and	third	one
0	0	1	1	1	0	0	1	0	1
1	0	1	0	1	0	2	1	0	1
2	1	0	0	0	1	0	1	1	0
3	0	1	1	1	0	0	1	0	1

Counts

words

```
cv = CountVectorizer(max_features=50000)
lr = LogisticRegression()
p = make_pipeline(cv, lr)
auc = cross_val_score(p, x, y,
    cv=3, scoring='roc_auc', n_jobs=-1)
np.mean(auc)
```

0.951354084277577

chars

```
cv = CountVectorizer(max_features=50000,
    analyzer='char')
p = make_pipeline(cv, lr)
auc = cross_val_score(p, x, y,
    cv=3, scoring='roc_auc', n_jobs=-1)
np.mean(auc)
```

0.805166242071356

TFIDF

tfidf score = tf x idf
tf - the number of times a term occurs in a given document
idf - number of documents in a corpus / number of documents that contain term

The goal of using tf-idf is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative

```
corpus = [
    'This is the first document.',
    'This is the second second document.',
    'And the third one.',
    'Is this the first document?',
]
```

```
model = TfidfVectorizer()
X = model.fit_transform(corpus)
pd.DataFrame(X.todense(),
             columns=model.vocabulary_)
```

executed in 17ms, finished 15:43:43 2018-03-22

	this	is	the	first	document	second	and	third	one
0	0.000000	0.438777	0.541977	0.438777	0.000000	0.000000	0.358729	0.000000	0.438777
1	0.000000	0.272301	0.000000	0.272301	0.000000	0.853226	0.222624	0.000000	0.272301
2	0.552805	0.000000	0.000000	0.000000	0.552805	0.000000	0.288477	0.552805	0.000000
3	0.000000	0.438777	0.541977	0.438777	0.000000	0.000000	0.358729	0.000000	0.438777

http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

TFIDF

words

```
tfidf = TfidfTransformer()
p = make_pipeline(cv, tfidf, lr)
auc = cross_val_score(p, x, y,
    cv=3, scoring='roc_auc', n_jobs=-1)
np.mean(auc)
```

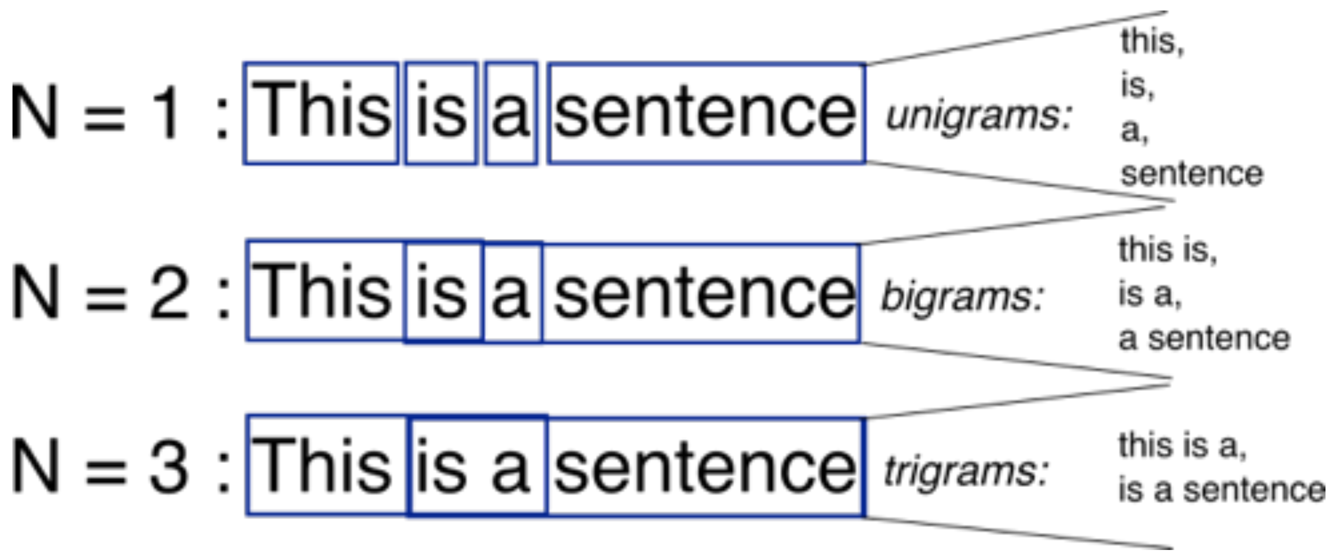
0.970447662337294

chars

```
cv = CountVectorizer(max_features=50000,
    analyzer='char')
p = make_pipeline(cv, tfidf, lr)
auc = cross_val_score(p, x, y,
    cv=3, scoring='roc_auc', n_jobs=-1)
np.mean(auc)
```

0.822682878743224

n-grams




```
corpus = [
    'This is the first document.',
    'This is the second second document.',
    'And the third one.',
    'Is this the first document?',
]
```

```
model = TfidfVectorizer(ngram_range=(2,2))
X = model.fit_transform(corpus)
pd.DataFrame(X.todense(),
             columns=model.vocabulary_)
```

executed in 23ms, finished 15:50:02 2018-03-22

	this is	is the	the first	first document	the second	second second	second document	and the	the third	third one	is this	this the
0	0.00000	0.500000	0.500000	0.000000	0.000000	0.000000	0.500000	0.000000	0.00000	0.00000	0.500000	0.000000
1	0.00000	0.000000	0.382743	0.000000	0.485461	0.485461	0.000000	0.485461	0.00000	0.00000	0.382743	0.000000
2	0.57735	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.57735	0.57735	0.000000	0.000000
3	0.00000	0.437791	0.000000	0.555283	0.000000	0.000000	0.437791	0.000000	0.00000	0.00000	0.000000	0.555283

n-grams

words

```
cv = CountVectorizer(max_features=50000,
        ngram_range=(1, 2))
tfidf = TfidfTransformer()
p = make_pipeline(cv, tfidf, lr)
auc = cross_val_score(p, x, y,
    cv=3, scoring='roc_auc', n_jobs=-1)
np.mean(auc)
```

0.9685204951401172

chars

```
cv = CountVectorizer(max_features=50000,
 analyzer='char', ngram_range=(3,5))
p = make_pipeline(cv, tfidf, lr)
auc = cross_val_score(p, x, y, cv=3,
    scoring='roc_auc', n_jobs=-1)
np.mean(auc)
```

0.972717694220589б

Naive Bayesian features

```
x = tfidf
r = np.log( P(y=1 | x) / P(y=0 | x ))
x = x * r
```

Select rows which belong to class 0
Calculate average of each column

Select rows which belong to class 1
Calculate average of each column

Divide average_0 / average_1
Multiply each row by resulting vector

https://nlp.stanford.edu/pubs/sidaw12_simple_sentiment.pdf

Naive Bayesian features

words

```
nb = NBFeaturer(1)
p = make_pipeline(cv, tfidf, nb, lr)
auc = cross_val_score(p, x, y,
    cv=3, scoring='roc_auc', n_jobs=-1)
np.mean(auc)
```

```
0.976195316977052
```

chars

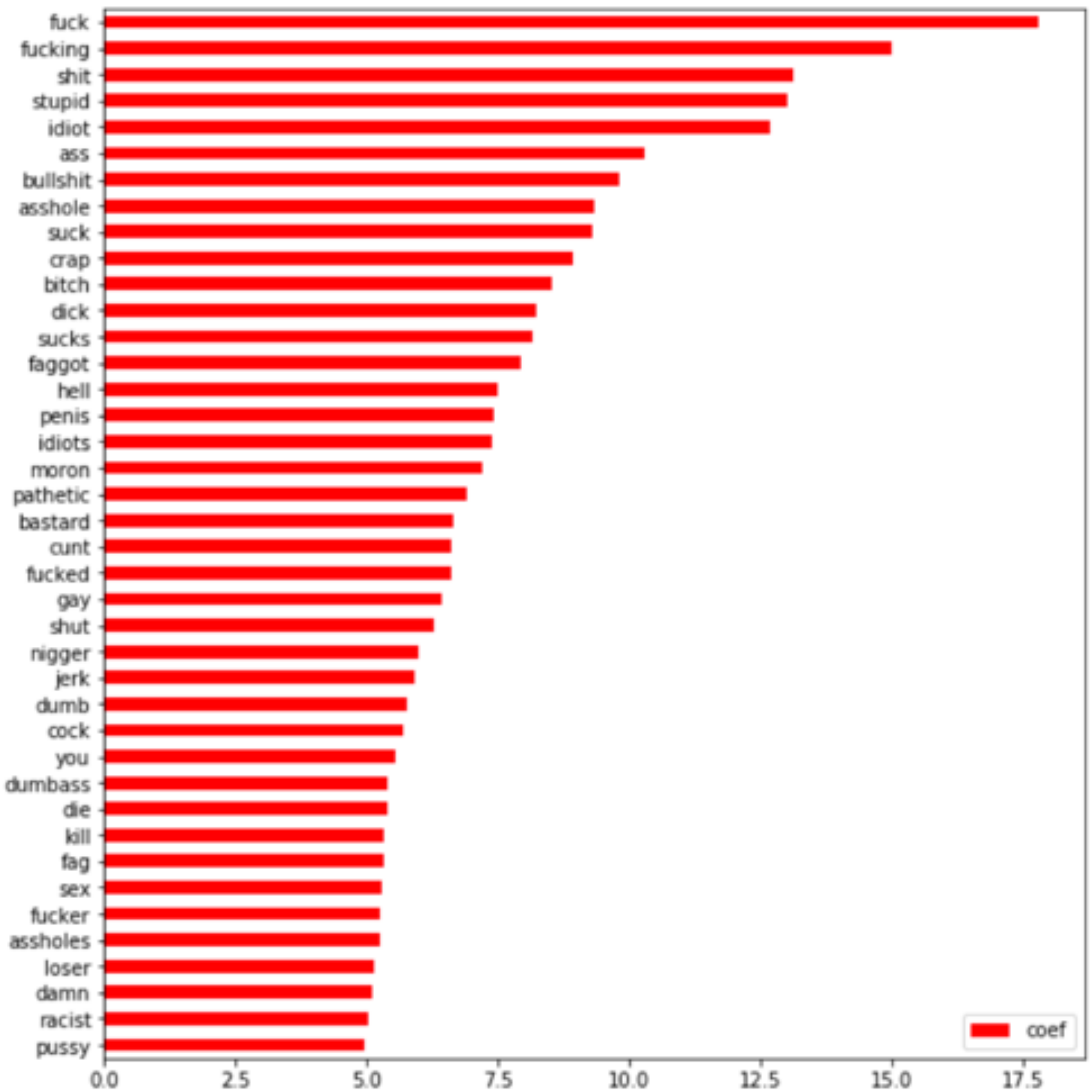
```
nb = NBFeaturer(1)
p = make_pipeline(cv, tfidf, nb, lr)
auc = cross_val_score(p, x, y, cv=3,
    scoring='roc_auc', n_jobs=-1)
np.mean(auc)
```

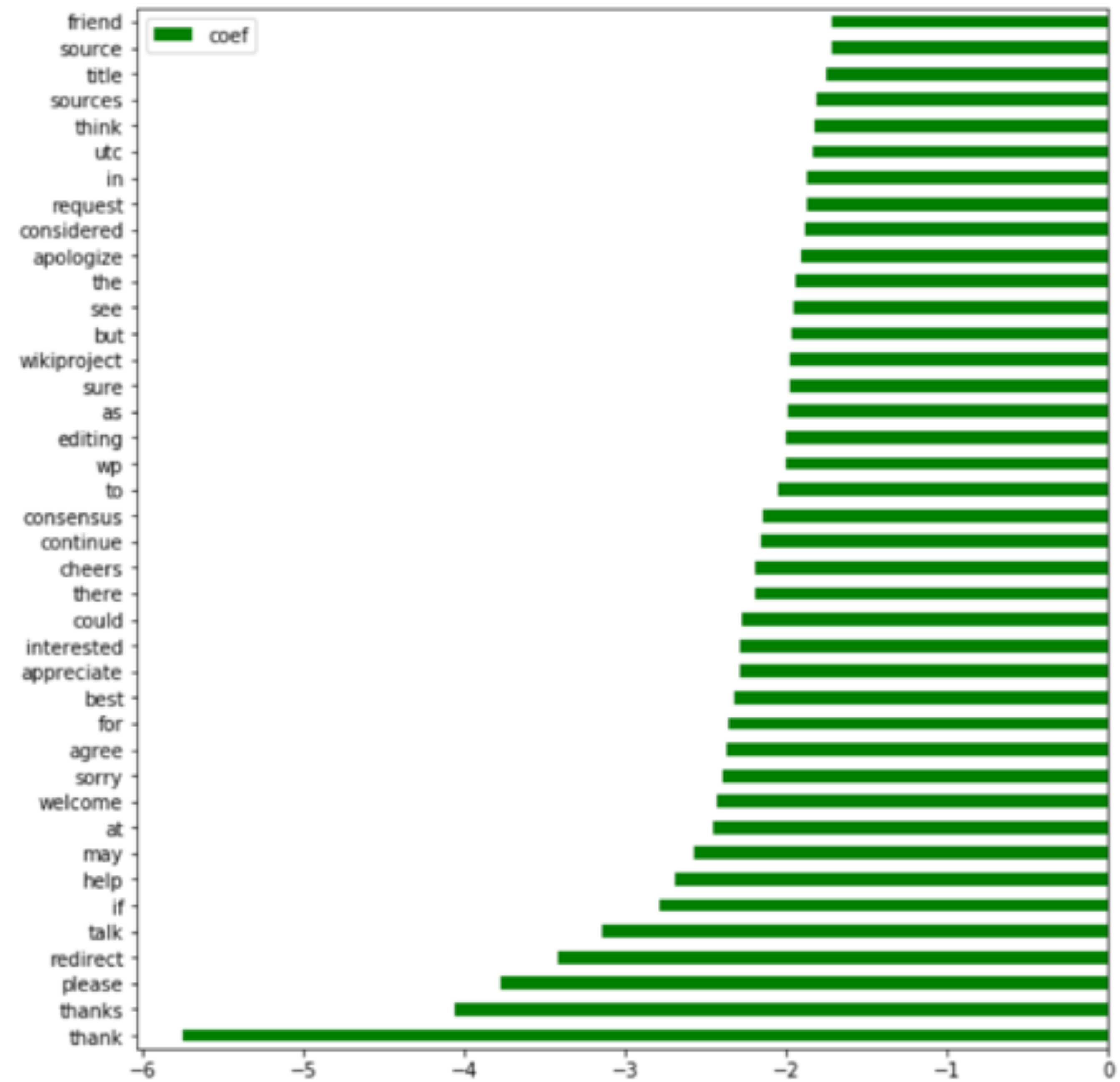
```
0.9758933057207932
```

Words polarity based on LR weights

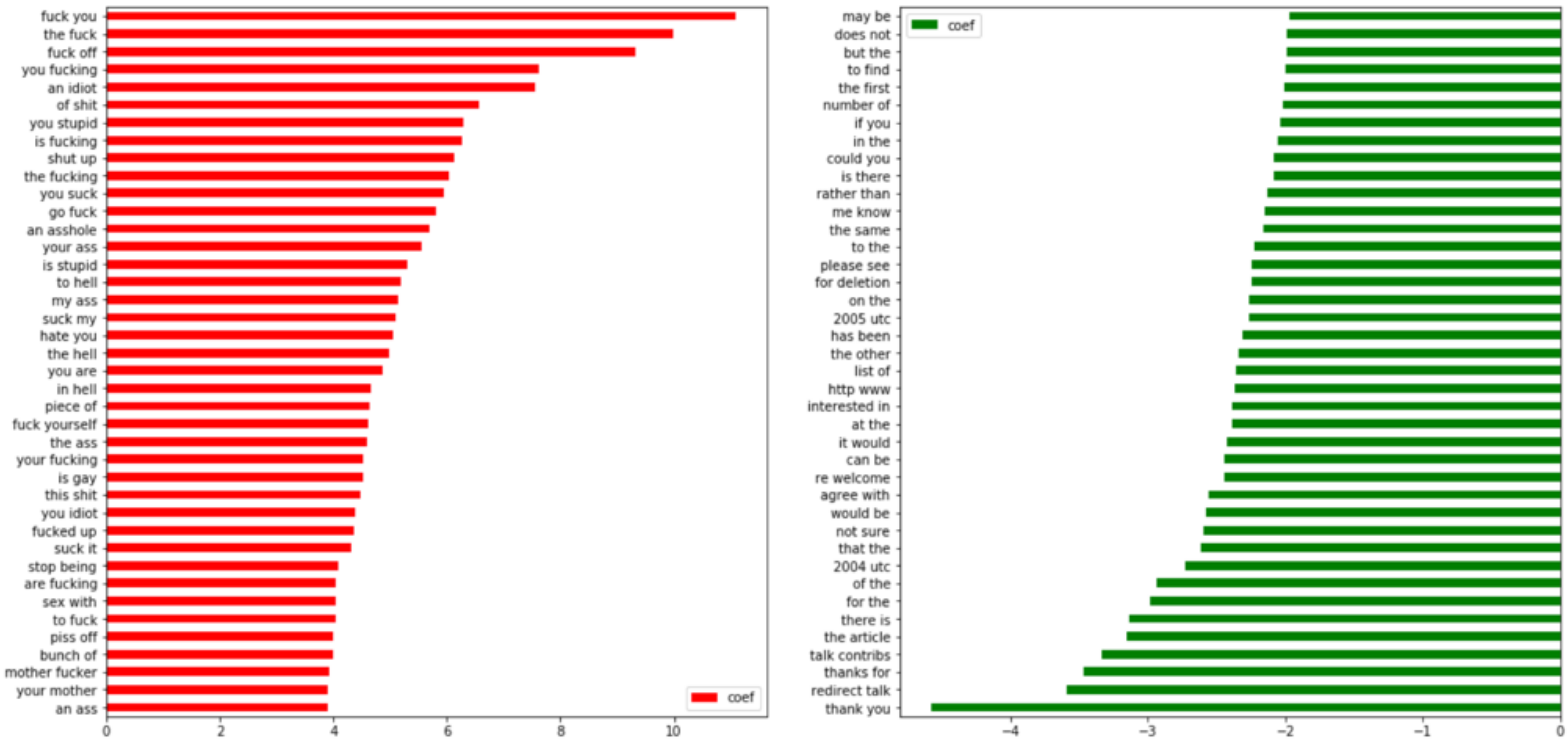
$$P(\mathbf{x}) = \frac{\exp z}{1 + \exp z},$$

$$z = \sum_{j=0}^{K} b_j x_j$$

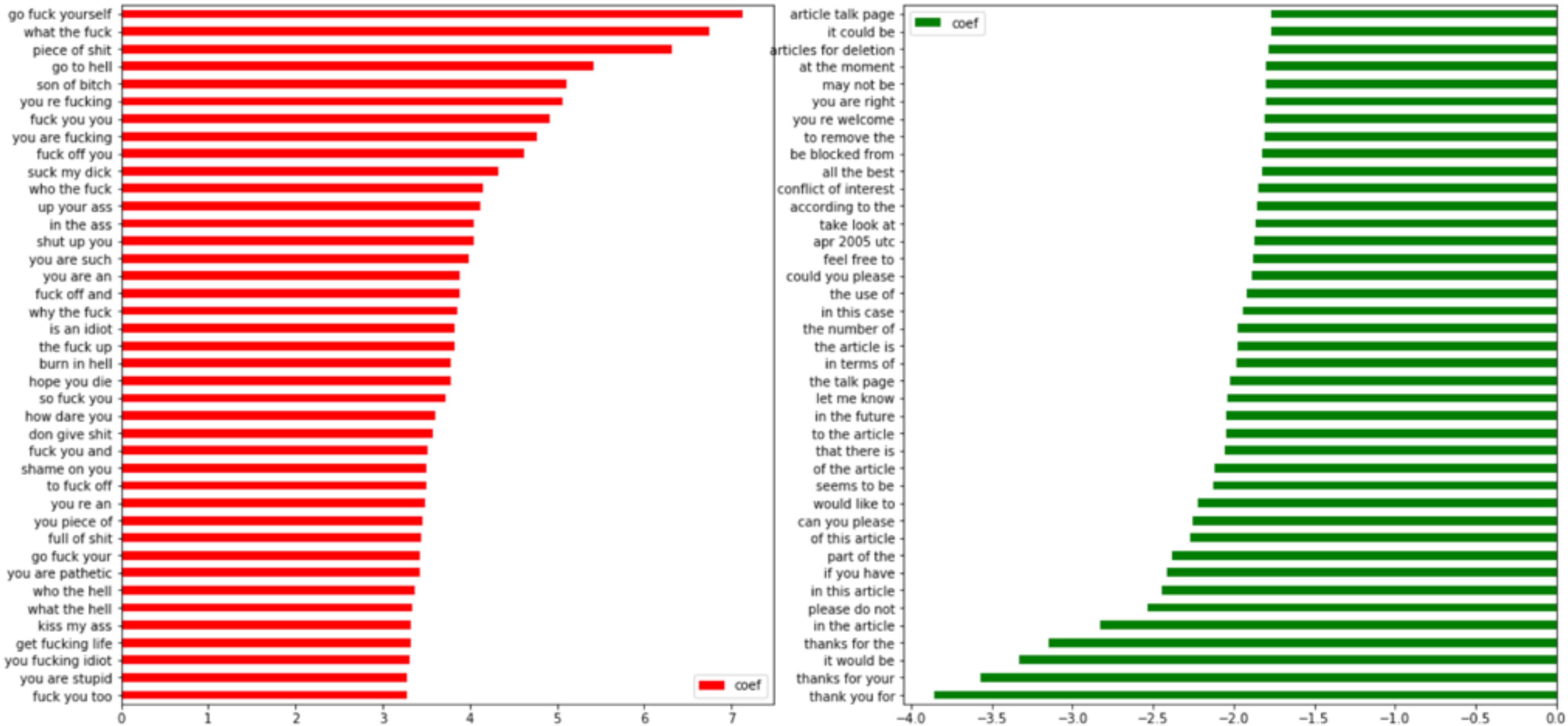




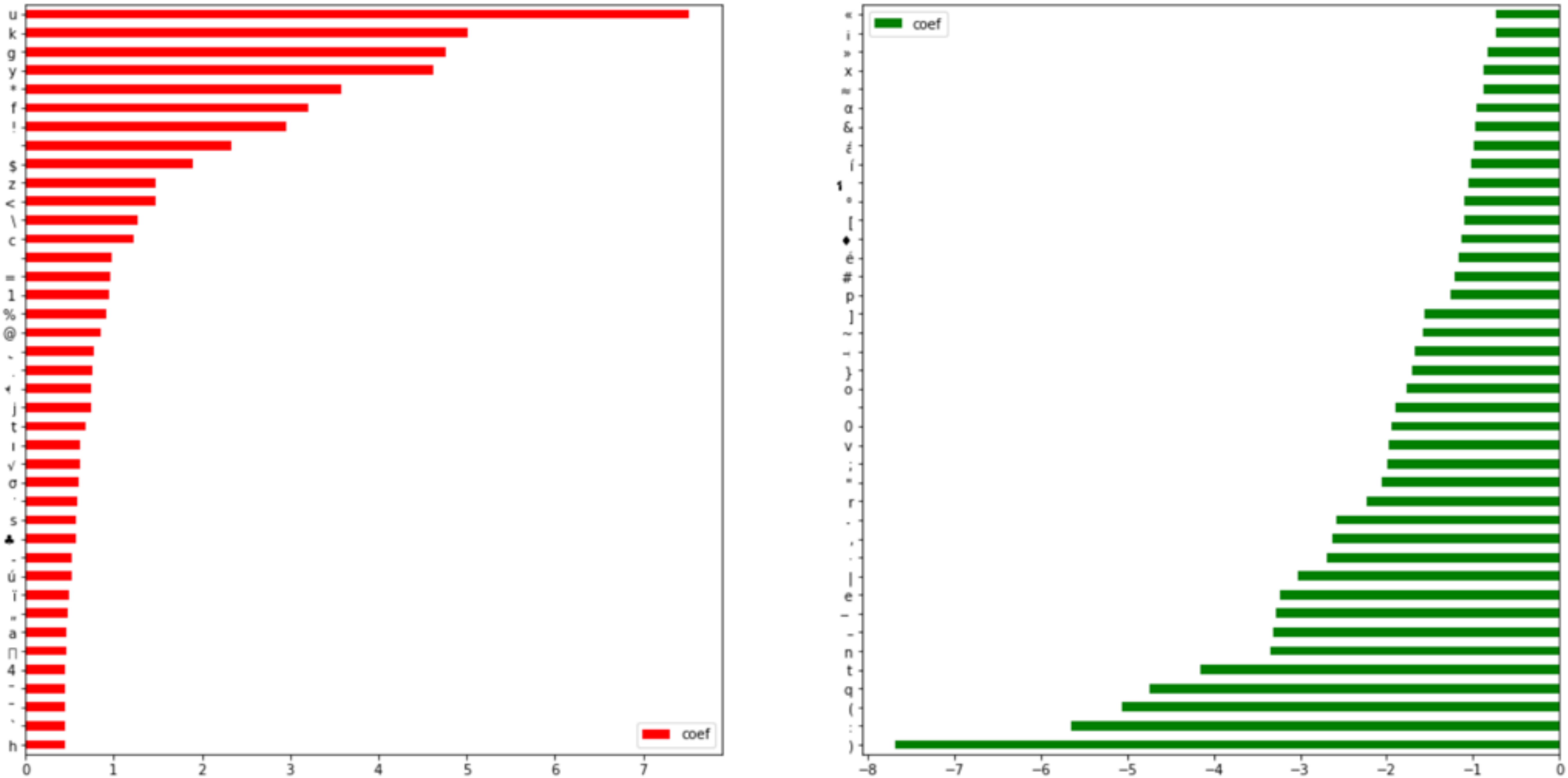
Words polarity based on LR weights, bi-grams



Words polarity based on LR weights, tri-grams



Words polarity based on LR weights, chars



Deep learning approach

- Pretrained words embeddings (word2vec)
- Data cleaning - minimize % of unknown tokens
- Data augmentation
- Apply model
  - RNN
  - GRU
  - LSTM
  - Other
    - CNN
    - HAN
    - DPCNN

One hot encoding vs Embedings

One hot encoding: every unique token gets its own binary vector, so if corpus contains 200 K tokens, resulting matrix is n x 200 000.

Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

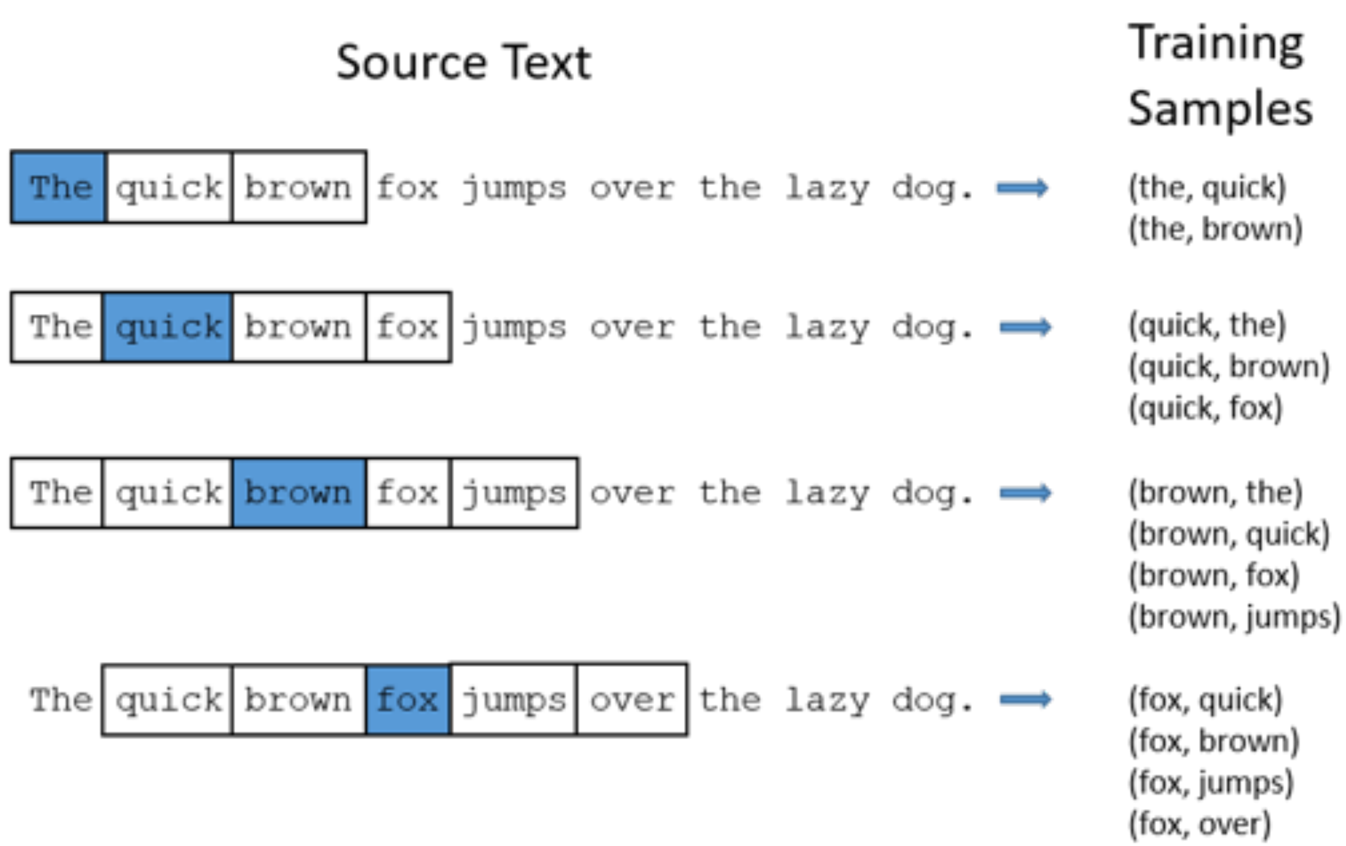
Embedings are just vectors with float numbers. To create unique vector we dont need to have number of columns equal to number of unique tokens. We can generate these vectors at random, feed them into neural network and learn what values should be in those vectors. Like parameters to learn.

Red	-0.99	1.05	0.05	…	0.12
Yellow	0.22	0.76	-0.88	…	-0.01
Green	-0.08	-0.02	-0.52	…	0.54

Word2vec

Given a specific word in the middle of a sentence (the input word), look at the words nearby and pick one at random. The network is going to tell us the probability for every word in our vocabulary of being the “nearby word” that we chose. "nearby", is actually a "window size" parameter to the algorithm. A typical window size might be 5, meaning 5 words behind and 5 words ahead (10 in total).

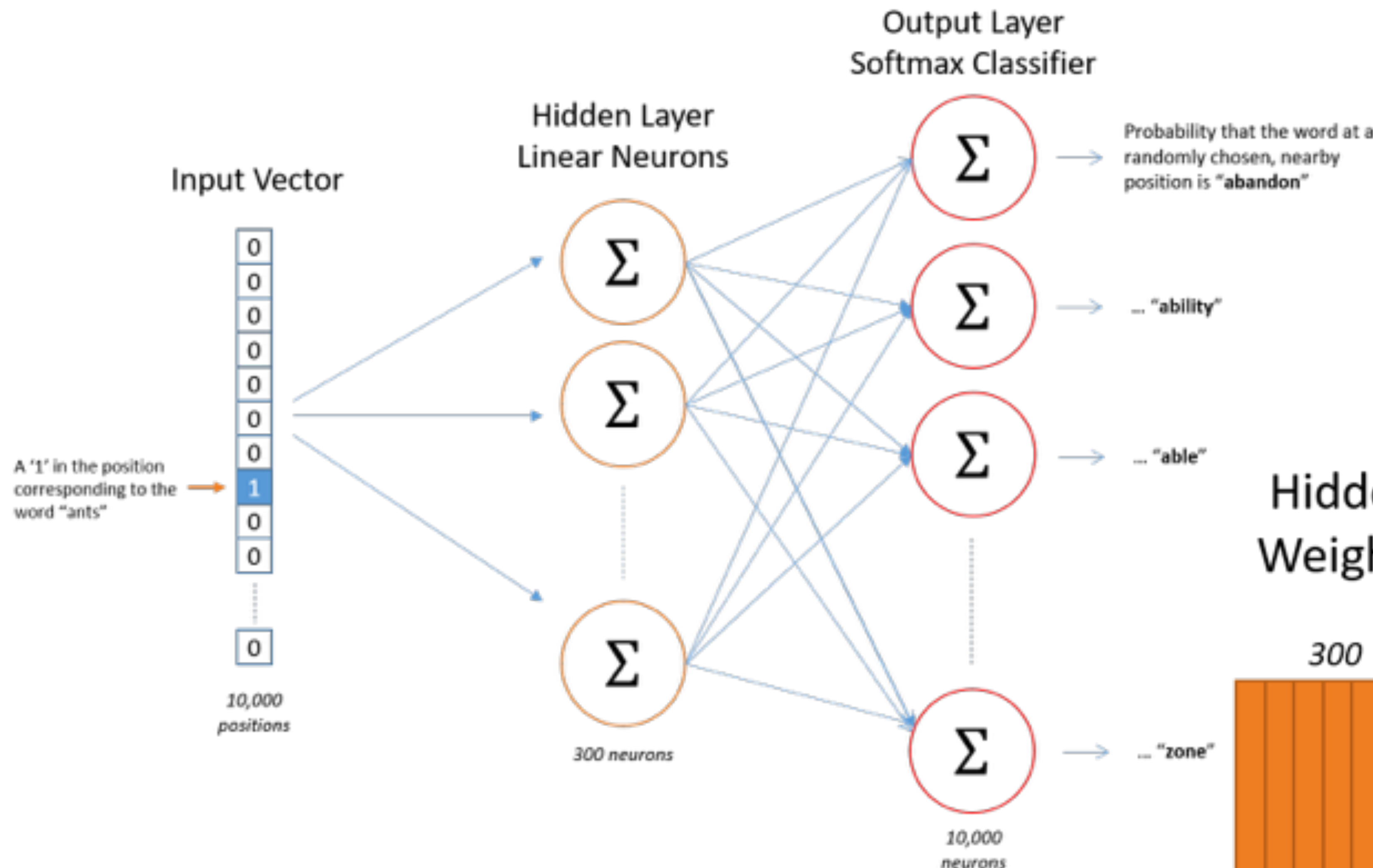
The output probabilities are going to relate to how likely it is find each vocabulary word nearby our input word.



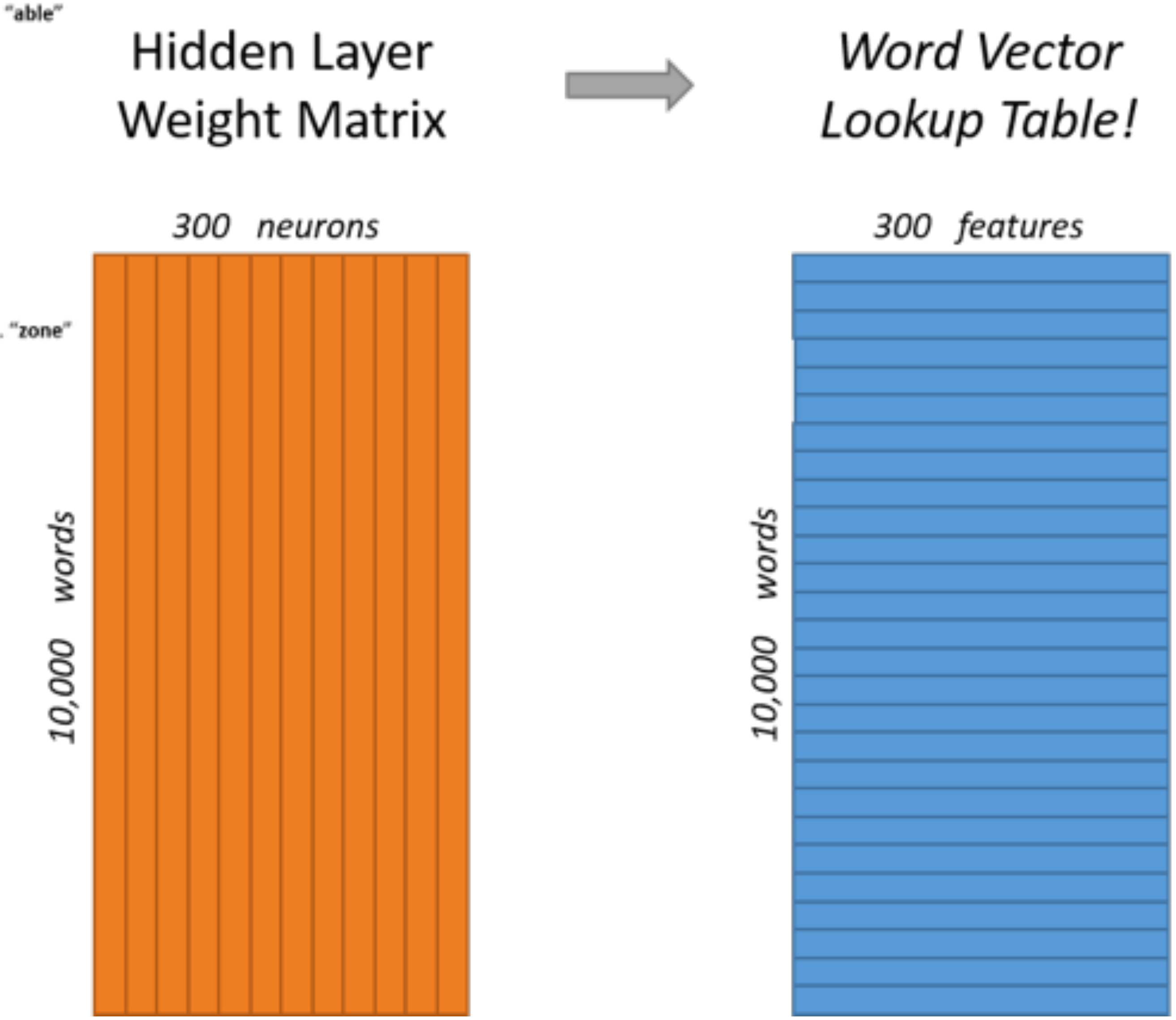
The network is going to learn the statistics from the number of times each pairing shows up

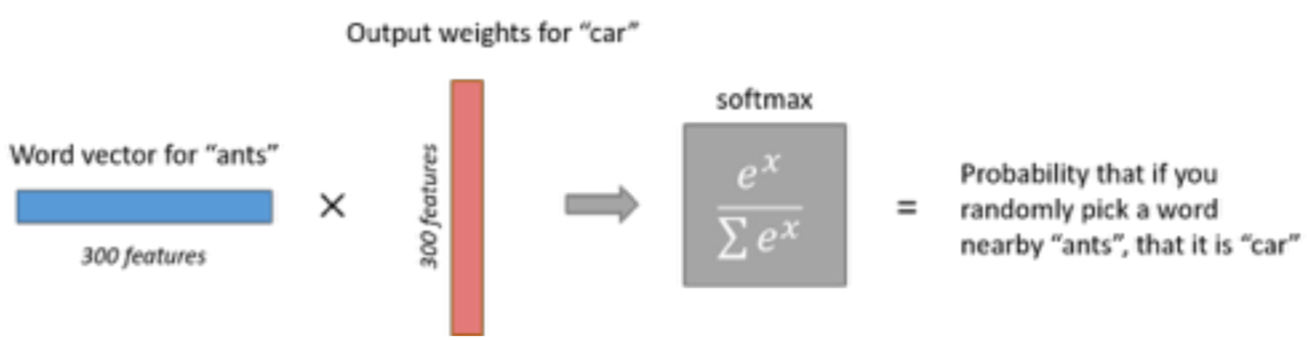
http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Word2vec



$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = \begin{bmatrix} 10 & 12 & 19 \end{bmatrix}$$

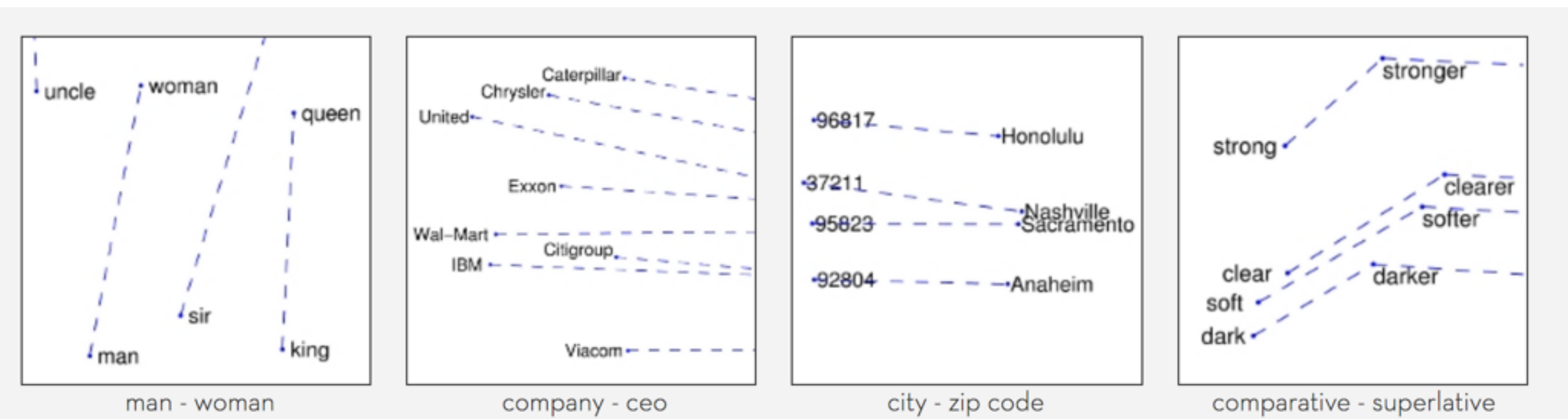




http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Word2vec

The underlying concept that distinguishes man from woman, i.e. sex or gender, may be equivalently specified by various other word pairs, such as king and queen or brother and sister. To state this observation mathematically, we might expect that **the vector differences man - woman, king - queen, and brother - sister might all be roughly equal**. This property and other interesting patterns can be observed in the above set of visualizations.



https://nlp.stanford.edu/projects/glove/

Glove, Fasttext

In order to compute word vectors, you need a large text corpus. Depending on the corpus, the word vectors will capture different information.

fastText

- 1 million word vectors trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16B tokens).
- 1 million word vectors trained with subword infomation on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16B tokens).
- 2 million word vectors trained on Common Crawl (600B tokens).

GloVe

Global Vectors for Word Representation

- Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
- Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
- Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
- Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip

https://fasttext.cc/docs/en/unsupervised-tutorial.html
https://nlp.stanford.edu/projects/glove/

Data cleaning - minimize % of unknown tokens

Issues:

- errors / typos **"FUCKK" : "FUCK"**
- intentional caps **"sUcks": “sucks" "suCks": "sucks"**
- hiden symbols **"f*ck": "fuck"**
- or even two **"f**k": "fuck"**
- word written together **'MOTHERFUCKERDIE': "motherfucker die"**
- overreaction **"MUAHAHAHAHAHAHAHAHAHAHAHAHAHAHAHAHAHA**

Data augmentation

Data augmentation with images: rotation, zoom, flip horizontal/vertical, stretching

Data augmentation with text: translation into another language and back

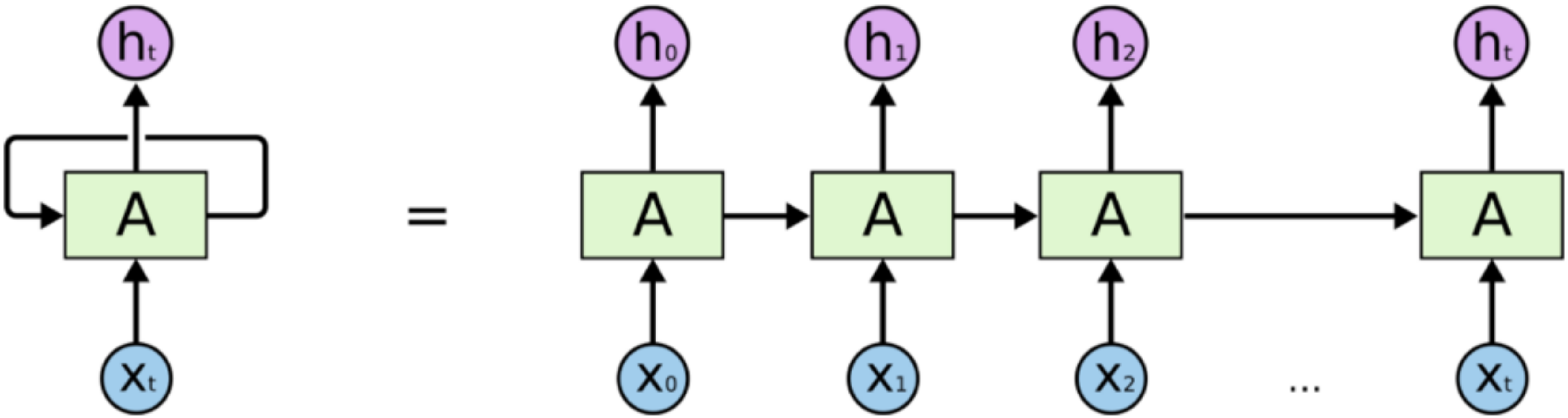
Original

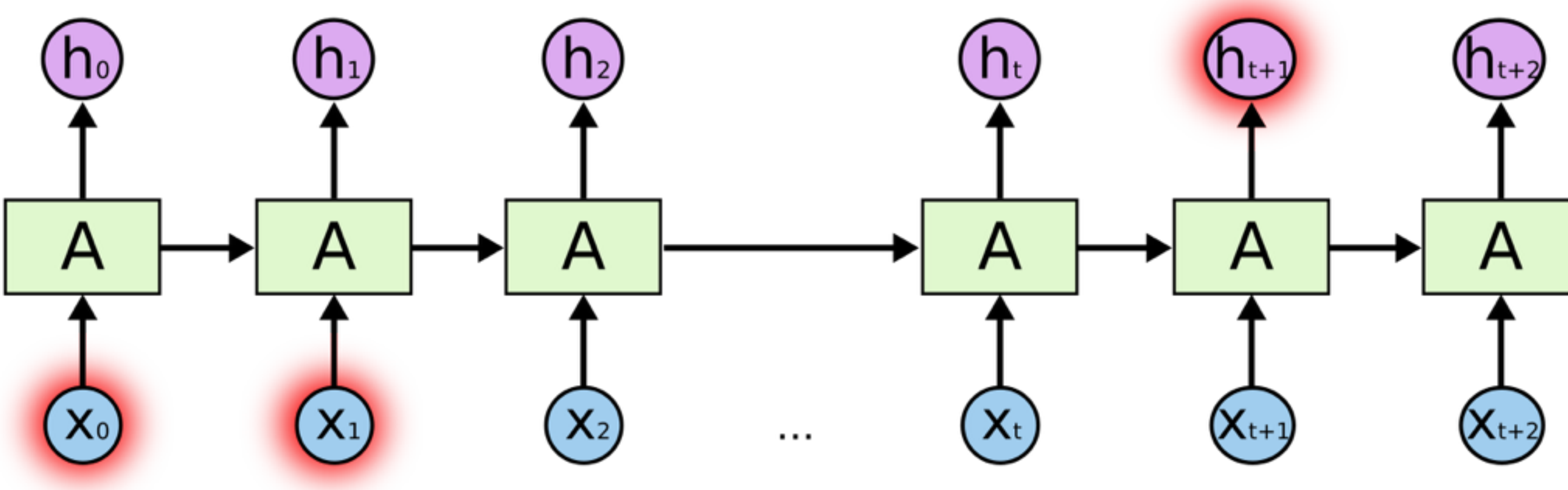
Wikipedia is full of fools. Who takes money and makes people work for free? Wikipedia!!! You might as well ban me, you fool. What's taking so long? Wiki is a stupid place, it's Jimbo's Cult.

EN -> DE -> EN

Wikipedia is full of dumbbells. Who takes money and lets people work for free? Wikipedia !!! You could ban me, you idiot. What does it take so long? Wiki is a stupid place, it's Jimbo's Cult.

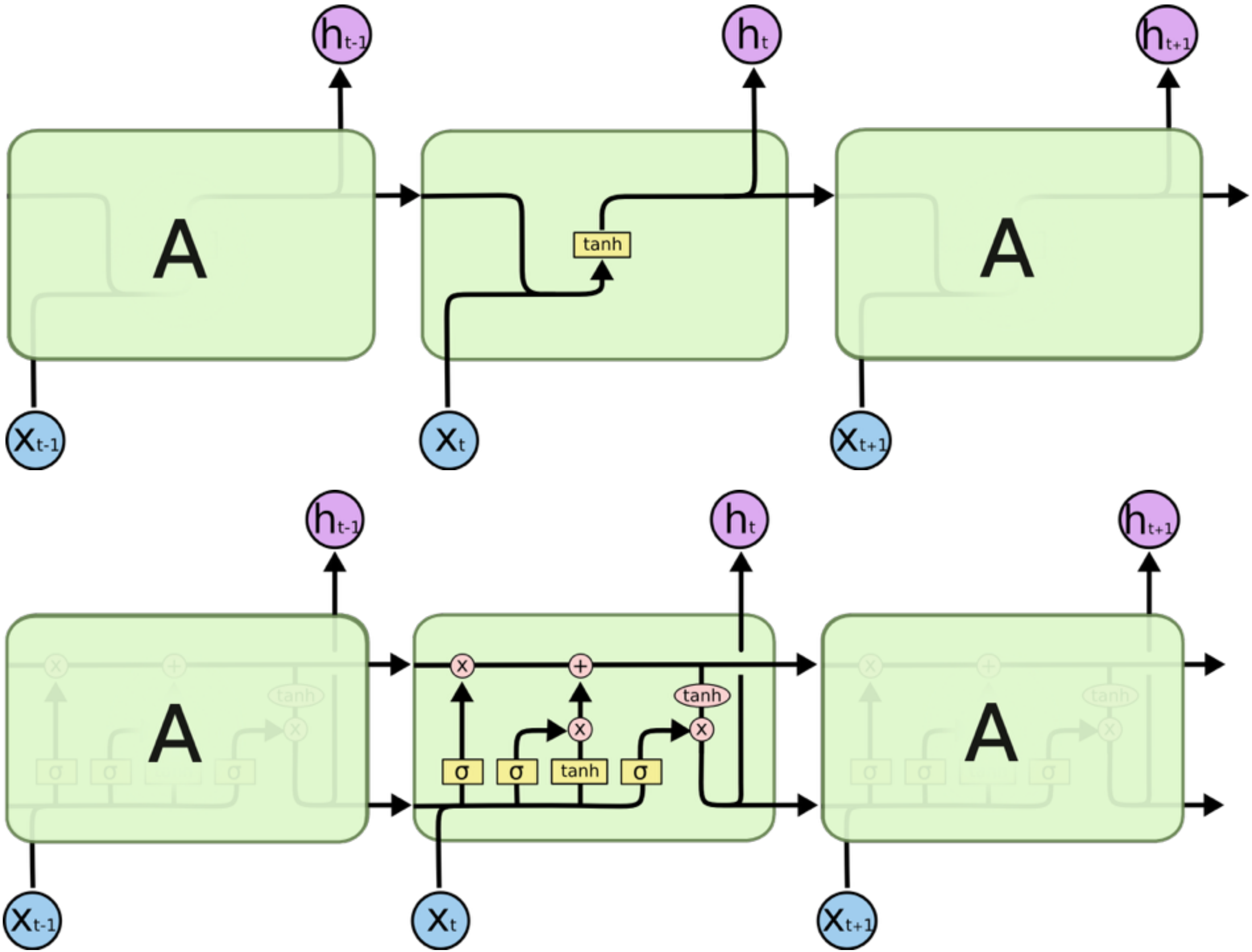
RNN





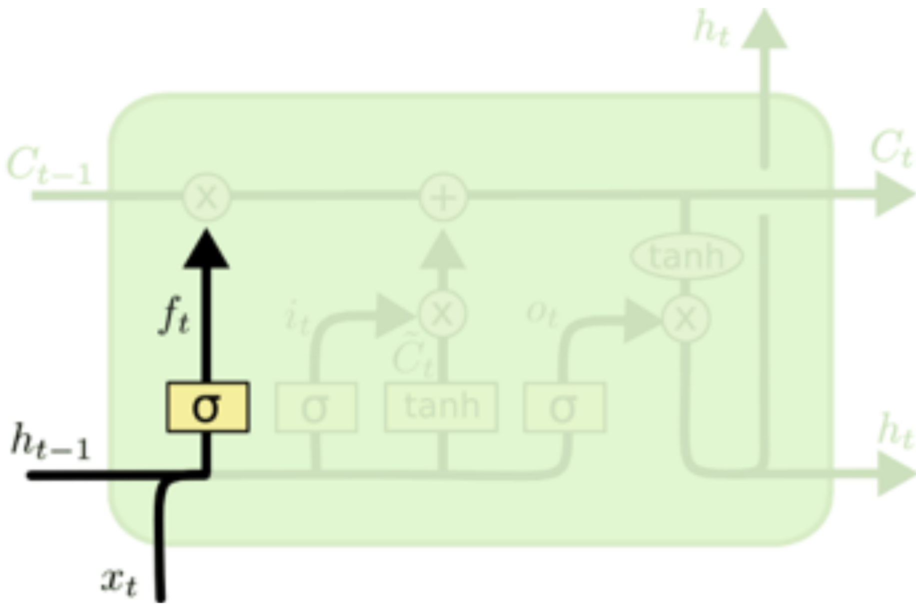
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN vs LSTM

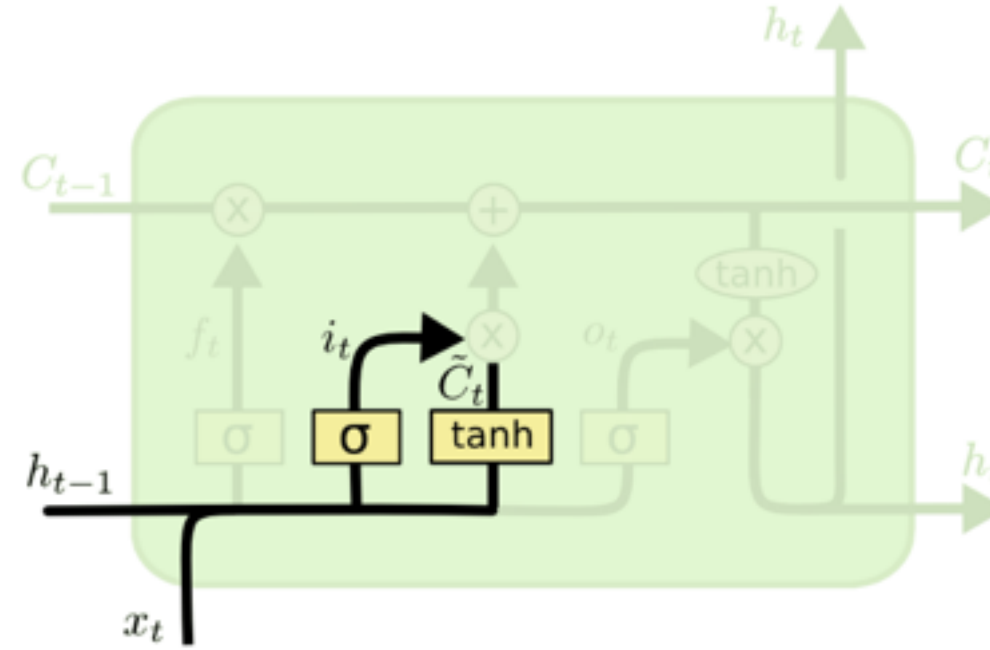


ht-1
ht
ht+1
A
tanh
A
Xt-1
Xt
Xt+1
ht-1
ht
ht+1
A
σ
σ
tanh
σ
tanh
A
Xt-1
Xt
Xt+1

LSTM

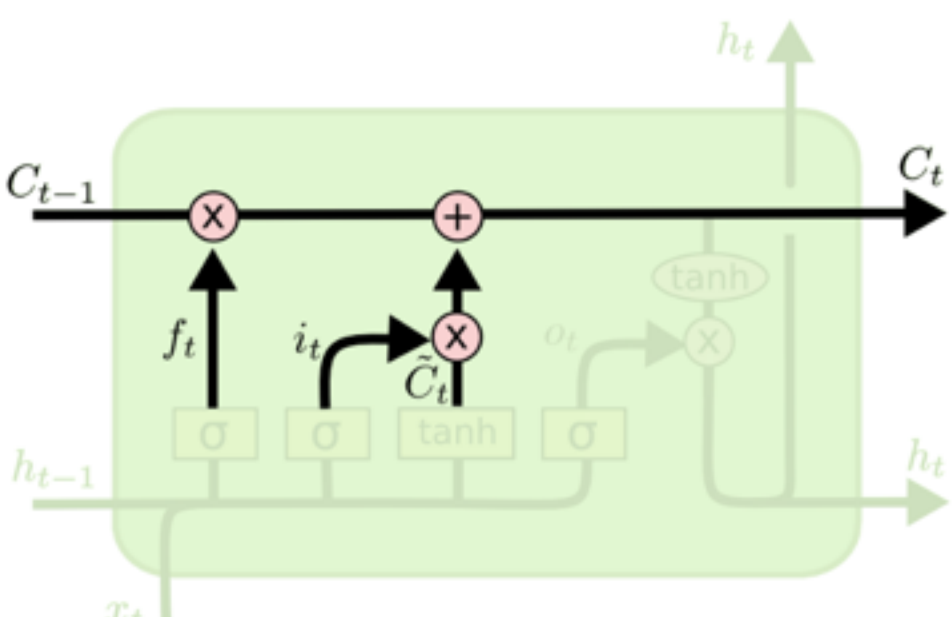


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

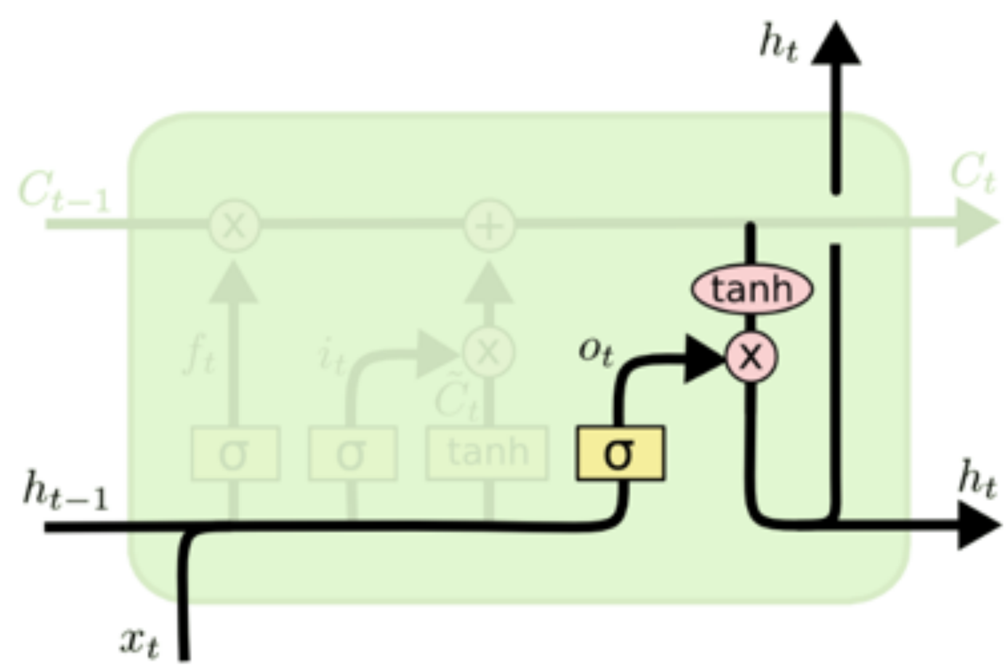


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$



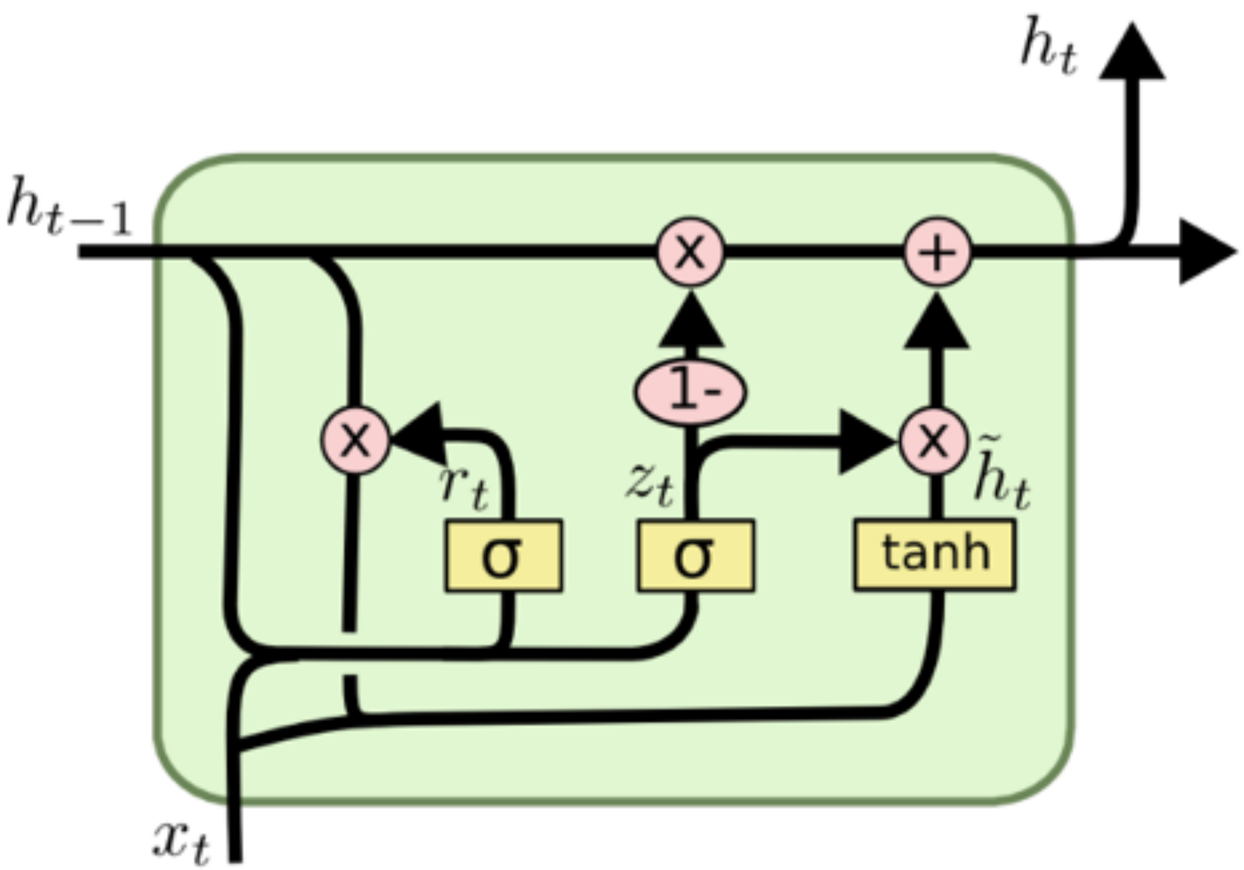
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

GRU



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

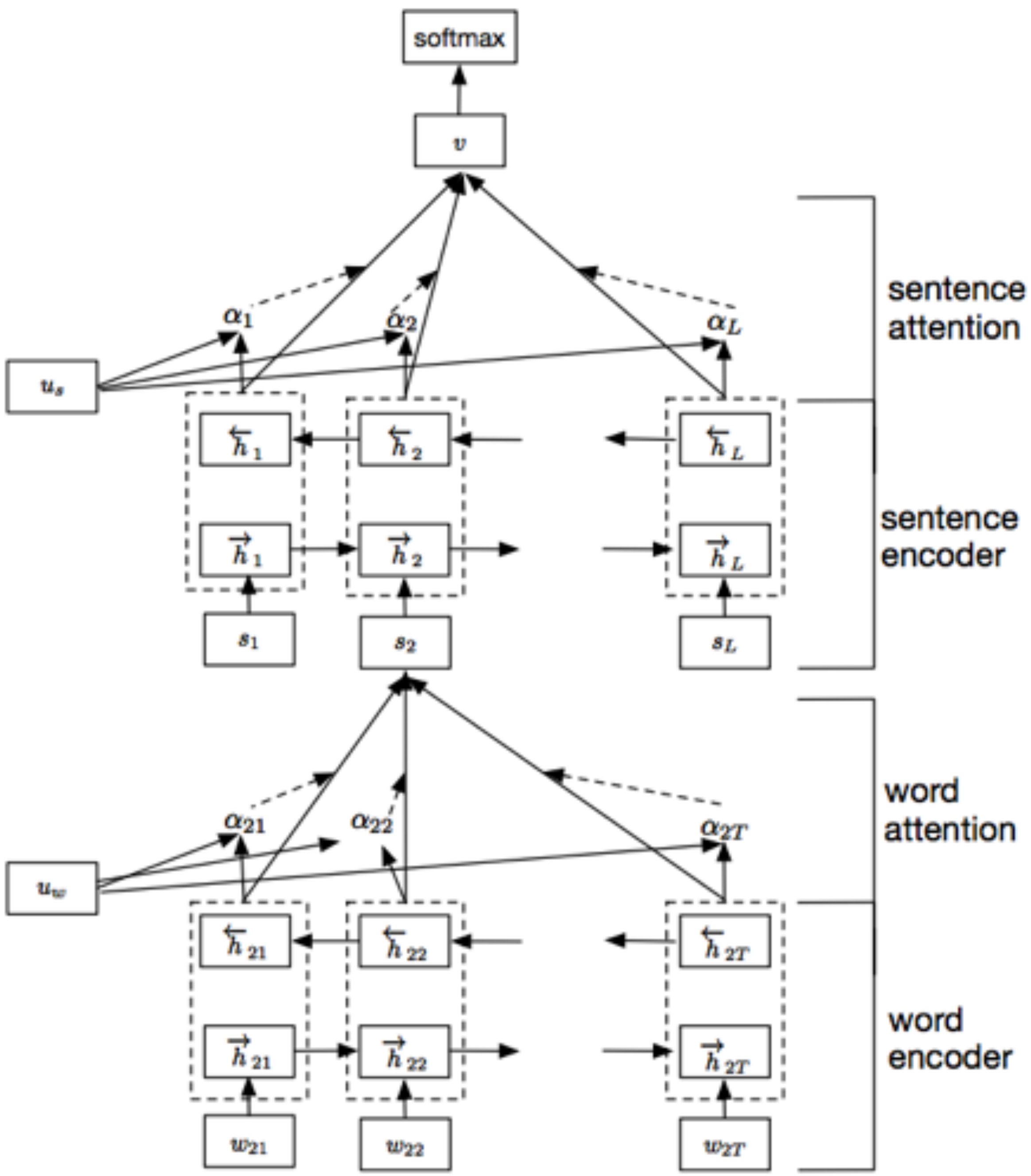
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

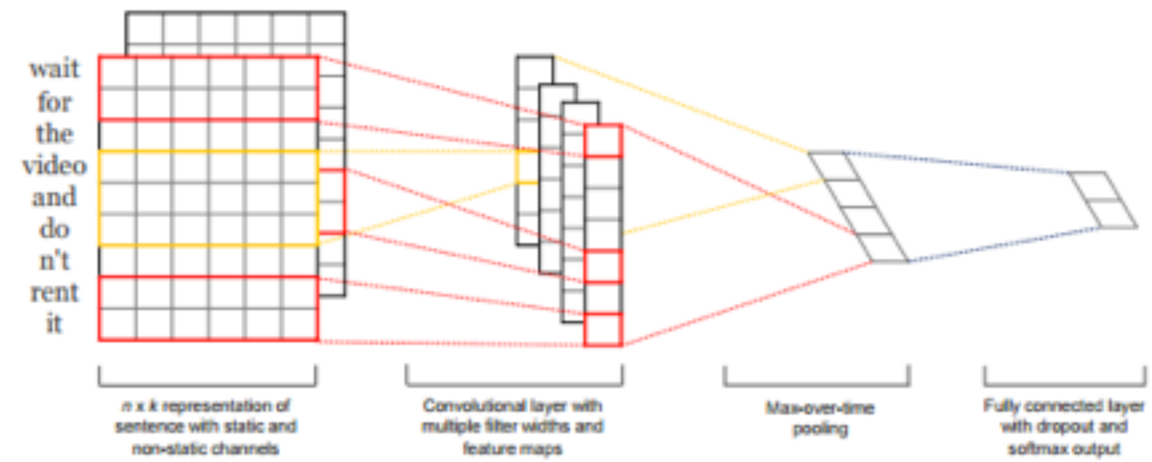
OTHER

Hierarchical Attention Network

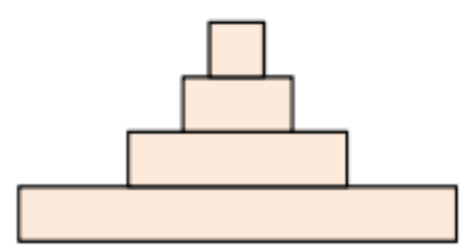


http://www.cs.cmu.edu/~./hovy/papers/16HLT-hierarchical-attention-networks.pdf

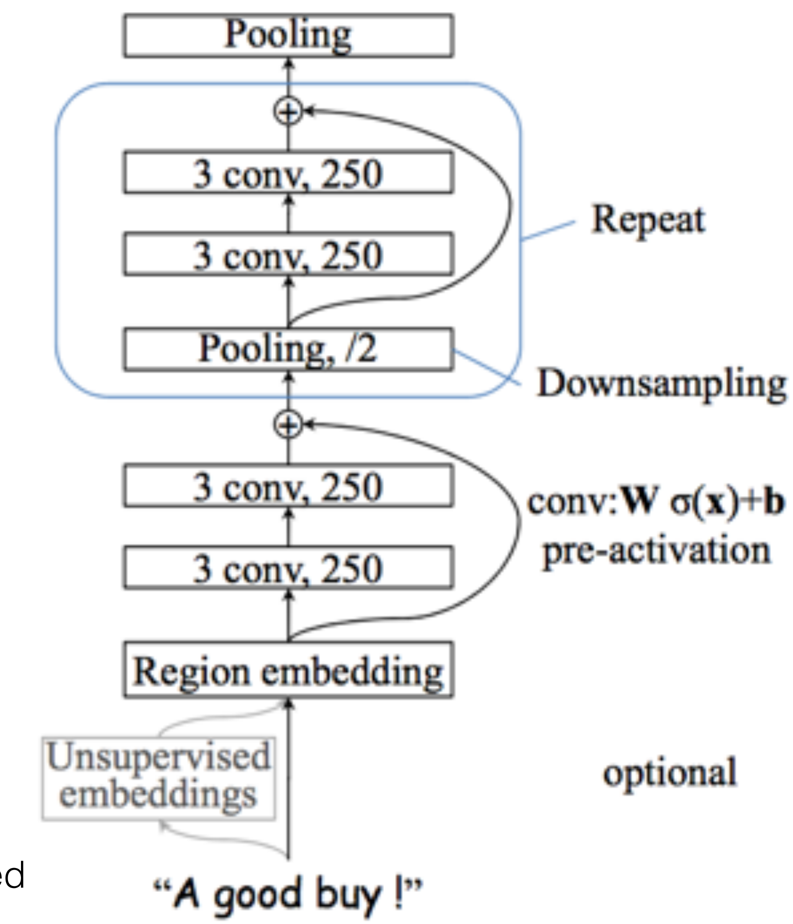
Convolutional Neural Network



Deep Pyramid Convolutional Neural Networks



Computation per layer is halved after every pooling.



(a) Our proposed model DPCNN

http://ai.tencent.com/ailab/media/publications/ACL3-Brady.pdf