1st Meetup on Machine Learning

Machine learning for text: fasttext

About me

I’m Matteo Pagliardini

Software / ML engineer at Iprova: http://www.iprova.com/

Using NLP to speed up the creation process.

Vector representations of chunks of text and classifiers are parts of our toolbox.

What is Fasttext ?

From the github:

“fastText is a library for efficient learning of word representations and sentence classification.”

Fasttext is fast / simple / accurate.

Guideline

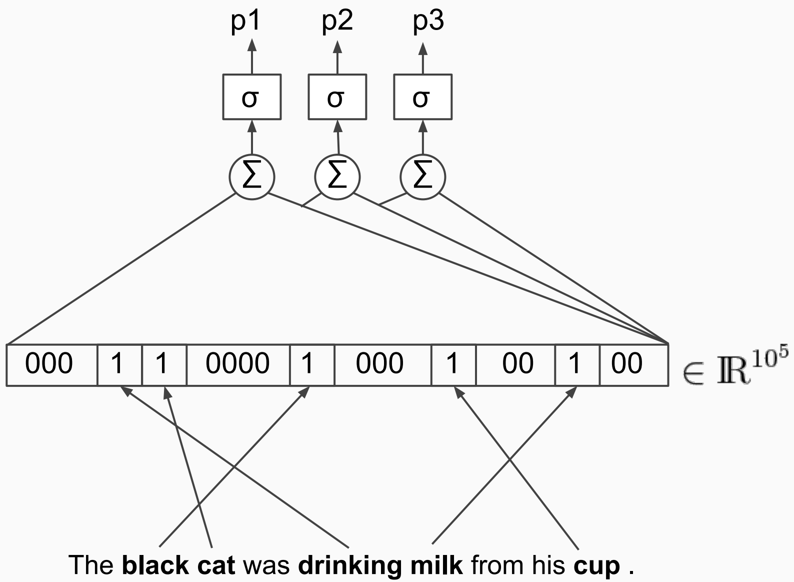
I. Fasttext for document classification

II. Fasttext for word embeddings

Fasttext for document classification

Standard baselines for text classification

- One simple baseline for text classification is to take a Bag of Words (BoW) representation and feed it to a linear classifier:



The pros:

- Convex optimization (global extrema)
- Simple (hyperplanes)

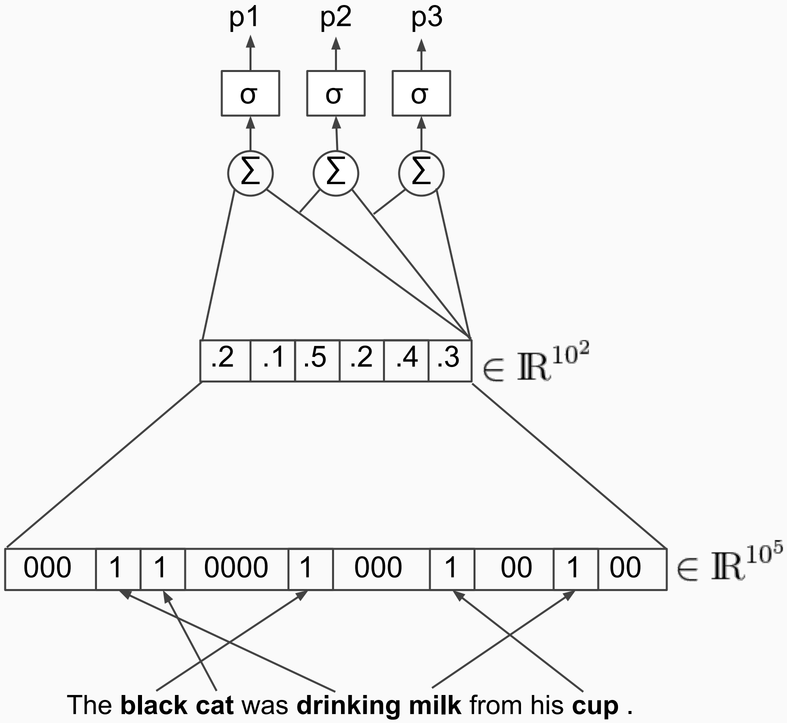
The cons:

- Each class is not learning from what other classes have learnt
- Easy to overfit for classes with few training examples

Solution: “factorize the linear classifier into low ranks matrices” aka add a hidden layer.

Adding low-rank constraints

- Adding low-rank constraints allows learning for one class to beneficiate other classes:



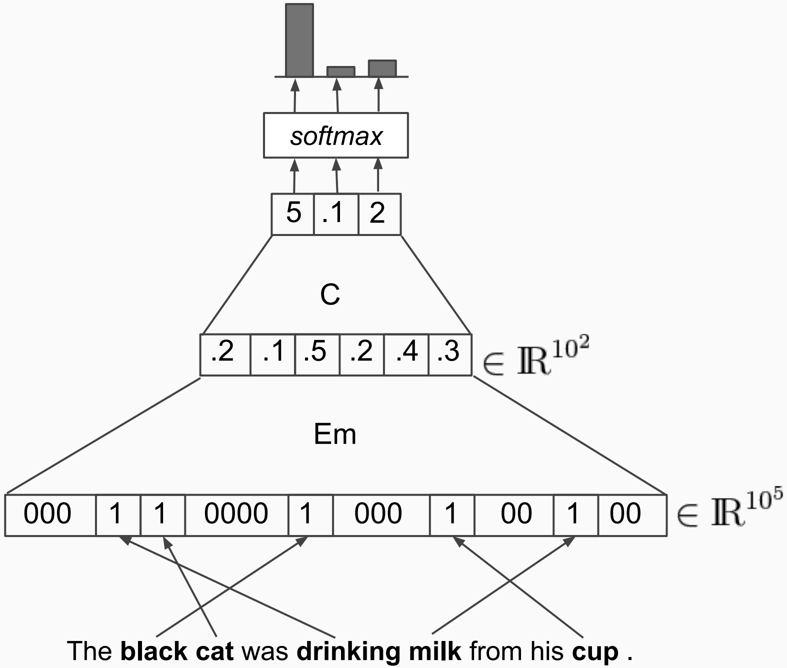
The model is now forcing the input representations to be squashed into dense representations. Those dense representations are obtained by the joined learning of all the classes together.

→ Better generalization for under-represented classes!

But this is not yet fasttext!

Getting closer to Fasttext

- Fasttext uses a softmax output layer instead of a logistic one:



The softmax output gives you a probability distribution over the labels instead of independent probabilities:

$$p_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

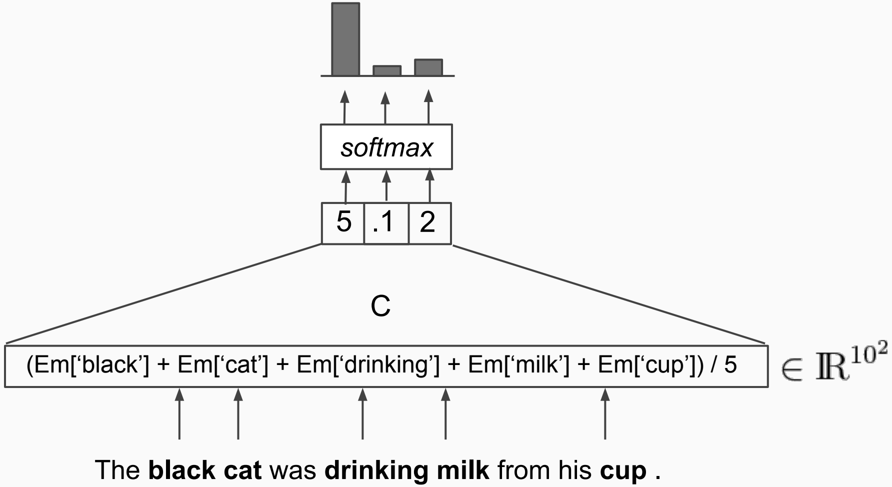
s is the vector of logits:

$$s = C.E_m.x_n$$

Em is the embedding matrix.
C is often referred to as the context matrix (we will see later why).

This is Fasttext

- Fasttext don’t just sum the embeddings of each input features, it averages them:


softmax
5
.1
2
C
(Em[‘black’] + Em[‘cat’] + Em[‘drinking’] + Em[‘milk’] + Em[‘cup’]) / 5
$\in \mathbb{R}^{10^2}$
The **black cat** was **drinking milk** from his **cup** .

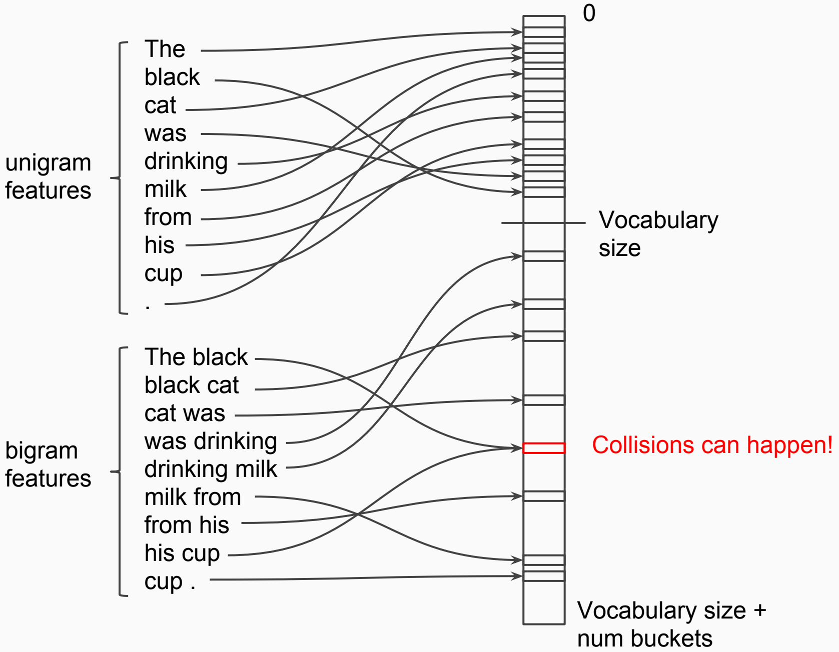
This is also exactly the Continuous Bag of Words CBoW word2vec architecture.

So what’s new ?

Fasttext features: the bag of tricks

- Fastext uses bags of n-grams:

Input sentence: *The black cat was drinking milk from his cup .*



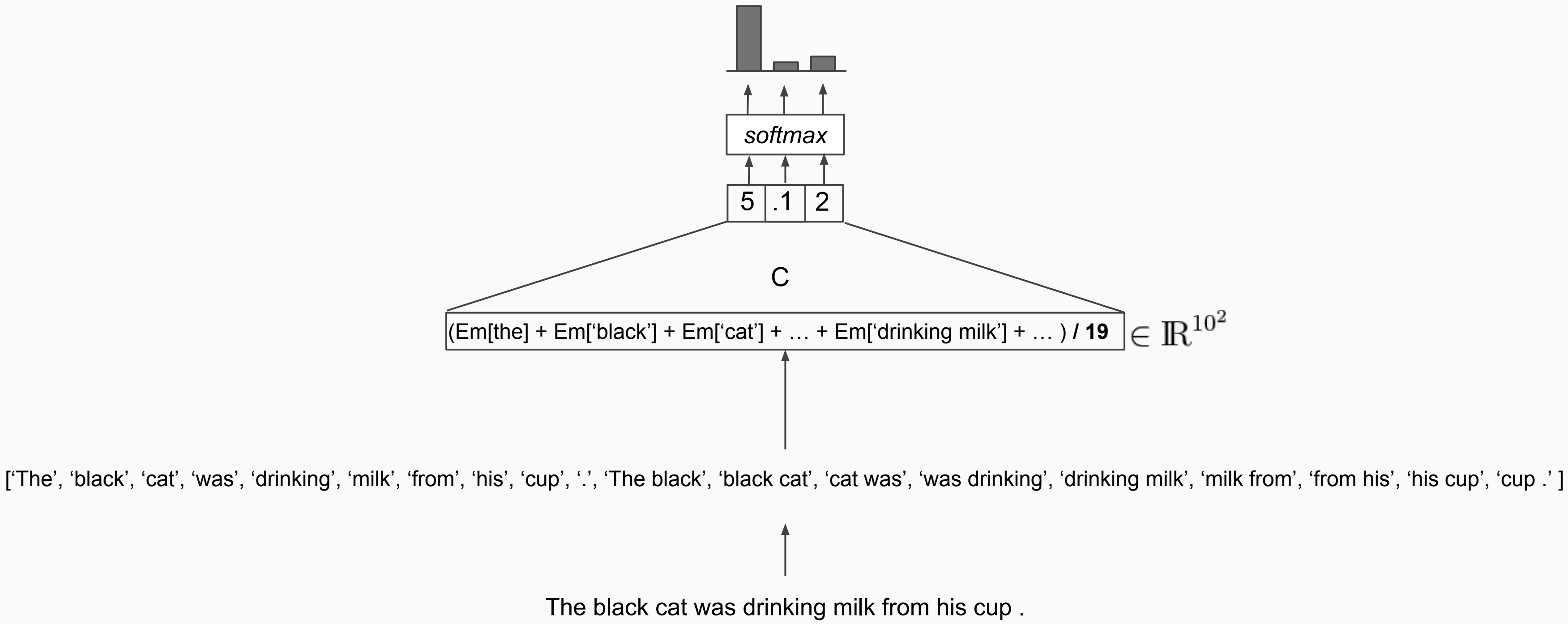
Working with n-grams can quickly yield important input dimensionality. The hashing trick used here allows us not to limit the input space size while still considering all the possible n-grams.

→ The price to pay is an input representation which is less flattened.

→ The smaller the number of buckets the more collisions.

Fasttext features: the bag of tricks

- In the end:



softmax

5 .1 2

C

(Em[the] + Em[‘black’] + Em[‘cat’] + … + Em[‘drinking milk’] + …) **/ 19** $\in \mathbb{R}^{10^2}$

[‘The’, ‘black’, ‘cat’, ‘was’, ‘drinking’, ‘milk’, ‘from’, ‘his’, ‘cup’, ‘.’, ‘The black’, ‘black cat’, ‘cat was’, ‘was drinking’, ‘drinking milk’, ‘milk from’, ‘from his’, ‘his cup’, ‘cup .’]

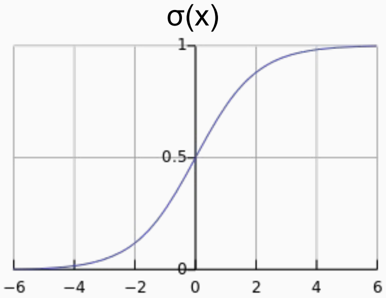
The black cat was drinking milk from his cup .

How is the training done?

- Some error metrics:

Mean squared error (MSE): $l = \frac{1}{N}\sum_{n=1}^{N}(\hat{y_n} - y_n)^2$ Where $\hat{y_n}$ is the true label.

Cross-Entropy (CE): $l = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{N_c}(\hat{y_n^c} ln(y_n^c) + (1 - \hat{y_n^c}) ln(1 - y_n^c)))$



The cross-entropy loss when we have a probability distribution as output with only one good class becomes the log-likelihood loss:

$$l = -\frac{1}{N}\sum_{n=1}^{N} ln(y_n^{tc})$$

Where *tc* is the good class

→ Fasttext attempts to learn a probability distribution over the classes, the log-likelihood loss is the natural choice.

How is the training done?

- The implementation uses pure stochastic gradient descent.

- The learning rate is decaying linearly to 0.

Speeding things up

- In almost all neural networks, there are two obvious bottlenecks:
  - The input layer: we often have discrete labels as input and the dimensionality can be large (10k - 1M).

    Often a fake issue. For the input, high dimensionality often means sparse data. And sparse data can be handled efficiently using sparse operations

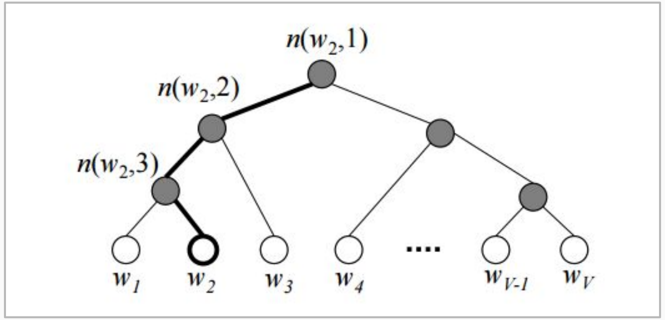
  - The output layer can have the same issues when we are doing unsupervised learning (input dim = output dim), or when we just have a lot of classes.

    Here it's more serious, to estimate the loss we need to compute the value for all the output classes.

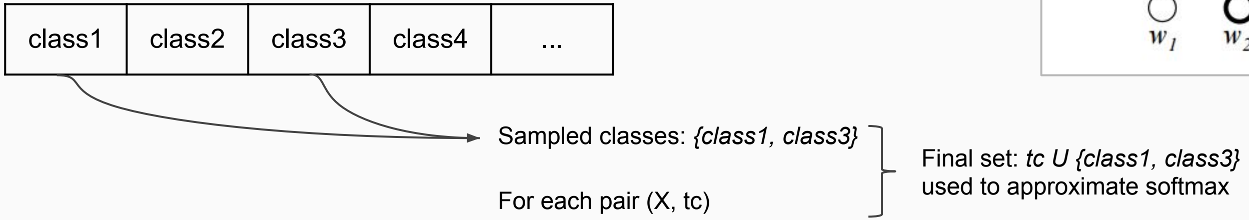
$$l = -\frac{1}{N}\sum_{n=1}^{N} ln(y_n^{tc})$$

Speeding things up

- There are three main ways to overcome this:
  - Hierarchical softmax:



  - Sampled softmax:



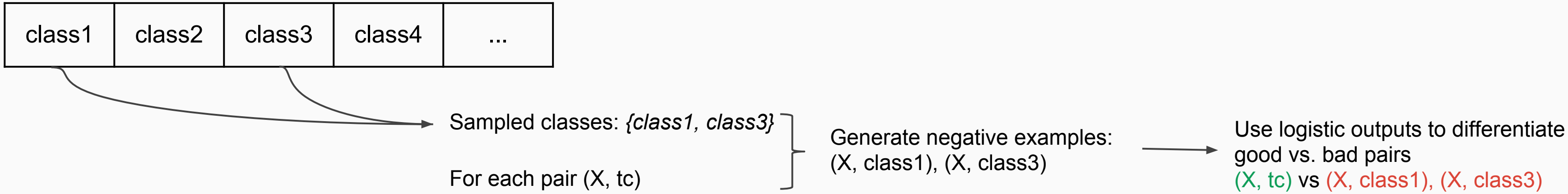
class1	class2	class3	class4	...

Sampled classes: *{class1, class3}*

For each pair (X, tc)

Final set: *tc U {class1, class3}* used to approximate softmax

  - Noise Contrastive Estimation (NCE):



class1	class2	class3	class4	...

Sampled classes: *{class1, class3}*

For each pair (X, tc)

Generate negative examples: (X, class1), (X, class3)

Use logistic outputs to differentiate good vs. bad pairs
(X, tc) vs (X, class1), (X, class3)

Summary of Fasttext for document classification

input
My graphic card is so hot right now.

pre-processing
my graphic card is so hot right now .

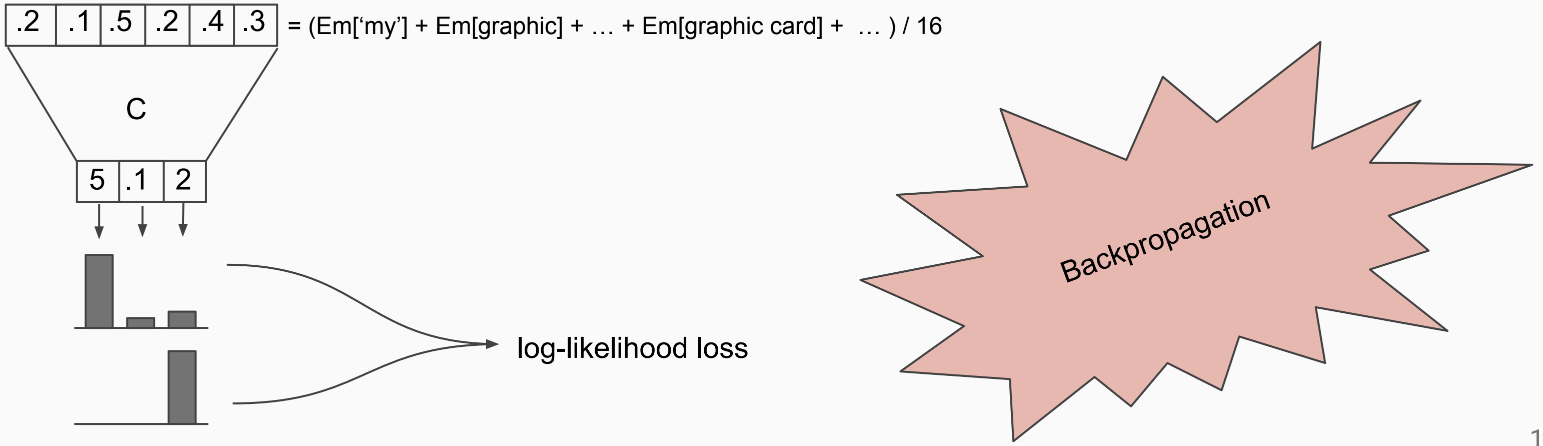
bag-of-tricks
[my, graphic, card, is, so, hot, right, now, my-graphic, graphic-card, card-is, is-so, so-hot, hot-right, right-now, now-.]

Avg of embeddings

logits

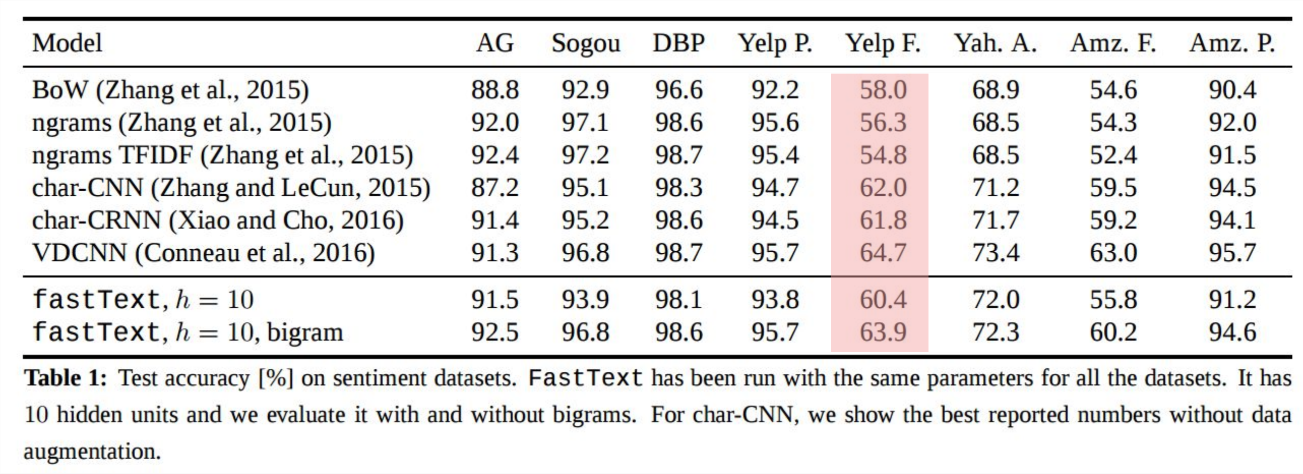
softmax predictions

True distribution



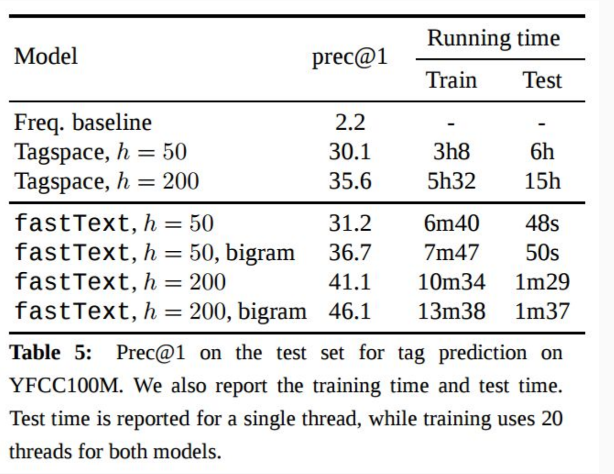
= (Em[‘my’] + Em[graphic] + … + Em[graphic card] + …) / 16

Paper's experiments



Model	AG	Sogou	DBP	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
BoW (Zhang et al., 2015)	88.8	92.9	96.6	92.2	58.0	68.9	54.6	90.4
ngrams (Zhang et al., 2015)	92.0	97.1	98.6	95.6	56.3	68.5	54.3	92.0
ngrams TFIDF (Zhang et al., 2015)	92.4	97.2	98.7	95.4	54.8	68.5	52.4	91.5
char-CNN (Zhang and LeCun, 2015)	87.2	95.1	98.3	94.7	62.0	71.2	59.5	94.5
char-CRNN (Xiao and Cho, 2016)	91.4	95.2	98.6	94.5	61.8	71.7	59.2	94.1
VDCNN (Conneau et al., 2016)	91.3	96.8	98.7	95.7	64.7	73.4	63.0	95.7
fastText, $h = 10$	91.5	93.9	98.1	93.8	60.4	72.0	55.8	91.2
fastText, $h = 10$, bigram	92.5	96.8	98.6	95.7	63.9	72.3	60.2	94.6

Table 1: Test accuracy [%] on sentiment datasets. FastText has been run with the same parameters for all the datasets. It has 10 hidden units and we evaluate it with and without bigrams. For char-CNN, we show the best reported numbers without data augmentation.



Model	prec@1	Running time	
		Train	Test
Freq. baseline	2.2	-	-
Tagspace, $h = 50$	30.1	3h8	6h
Tagspace, $h = 200$	35.6	5h32	15h
fastText, $h = 50$	31.2	6m40	48s
fastText, $h = 50$, bigram	36.7	7m47	50s
fastText, $h = 200$	41.1	10m34	1m29
fastText, $h = 200$, bigram	46.1	13m38	1m37

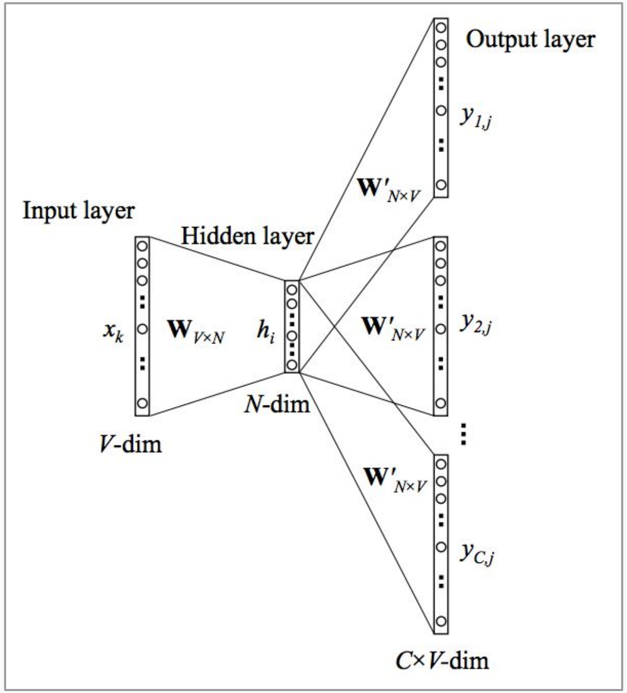
Table 5: Prec@1 on the test set for tag prediction on YFCC100M. We also report the training time and test time. Test time is reported for a single thread, while training uses 20 threads for both models.

Coding supervised fasttext

Fasttext for word vectors

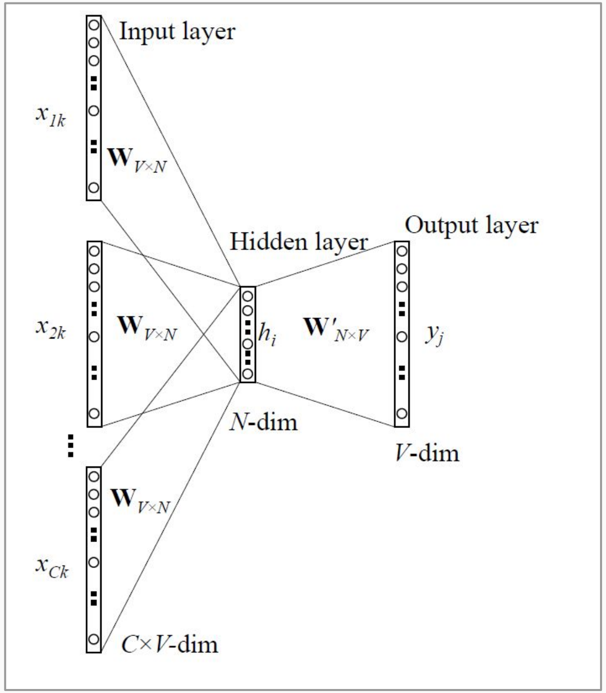
The big picture

- Most of you are probably familiar with word2vec:



Skip-gram
p(context | word)

$$\text{When a } \underbrace{\text{cat } \overset{\downarrow}{\text{chases}} \text{ its prey}}_{\text{Context window}}\text{, it keeps its head level.}$$



CBoW
p(word | context)

The big picture

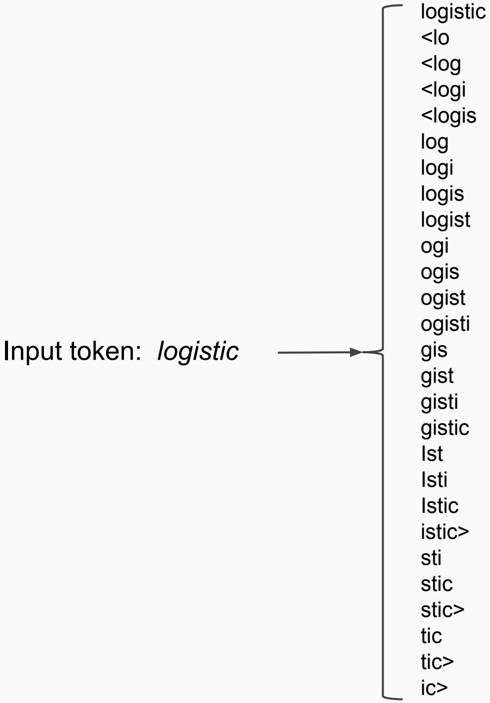
- Word2vec does not consider morphology in the construction of its embeddings:
  - Some languages are morphologically rich e.g. German. Higher morphology often goes with less overlaps at the token level.

    Kraftfahrzeughaftpflichtversicherung → *automobile liability insurance*

  - Some rare words could have their sense inferred from their morphology (e.g. technical terms)
  - Spelling errors in the corpus can prevent ourselves to find all the possible contexts of an entity.

Characters n-grams

- To consider morphology, fasttext extract all the character n-grams of size 3 to 6:



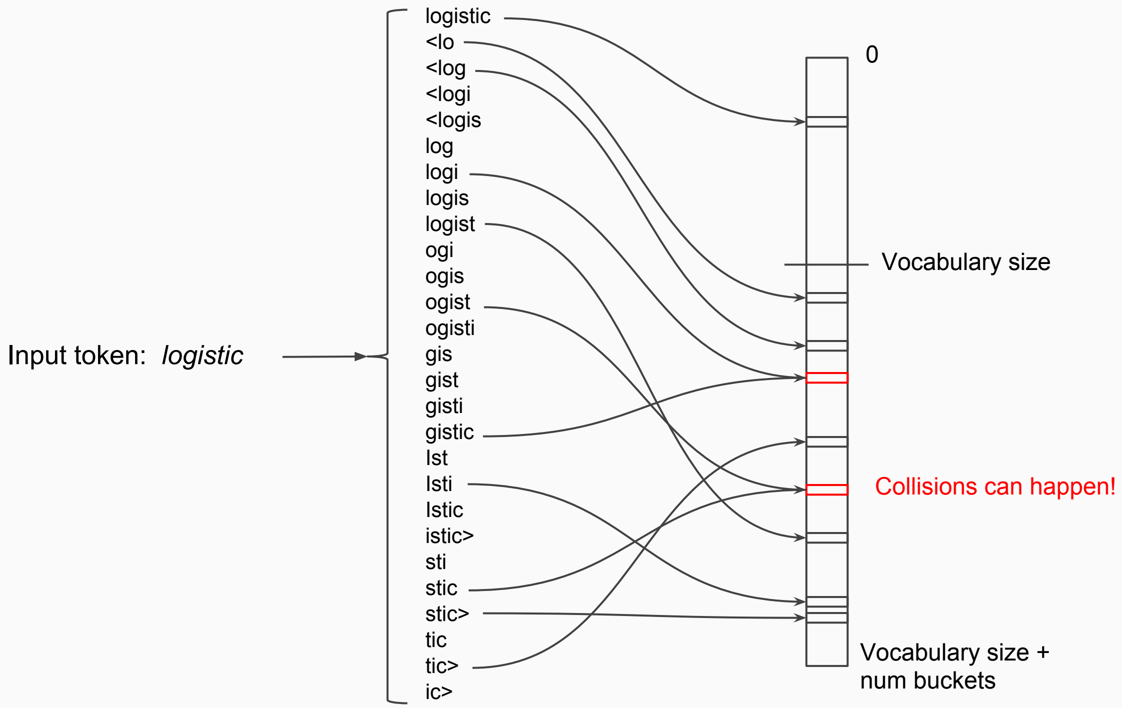
Input token: *logistic*

logistic
<lo
<log
<logi
<logis
log
logi
logis
logist
ogi
ogis
ogist
ogisti
gis
gist
gisti
gistic
Ist
Isti
Istic
istic>
sti
stic
stic>
tic
tic>
ic>

The tokens < and > are added to differentiate prefixes from suffixes

The word itself (unigram) is also considered by the algorithm.

The bag of tricks all over again



Input token: *logistic*

logistic
<lo
<log
<logi
<logis
log
logi
logis
logist
ogi
ogis
ogist
ogisti
gis
gist
gisti
gistic
lst
lsti
lstic
istic>
sti
stic
stic>
tic
tic>
ic>

0

Vocabulary size

Collisions can happen!

Vocabulary size +
num buckets

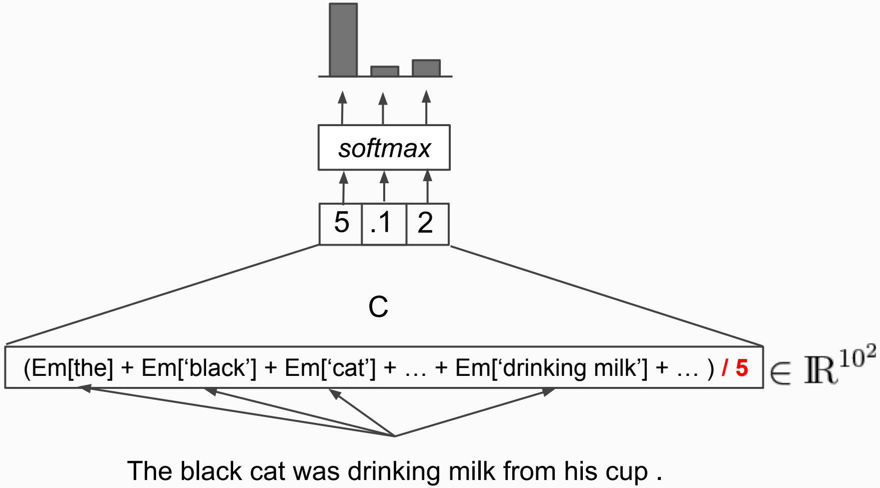
The unigrams are mapped to [0, vocab size[

The char n-grams are mapped to:
[vocab size, Vocab size + num buckets]

Collisions can exist between characters n-grams.

From CBoW to Skip-gram: a question of perspective

- The architecture of supervised and unsupervised fasttext are almost the same:



softmax

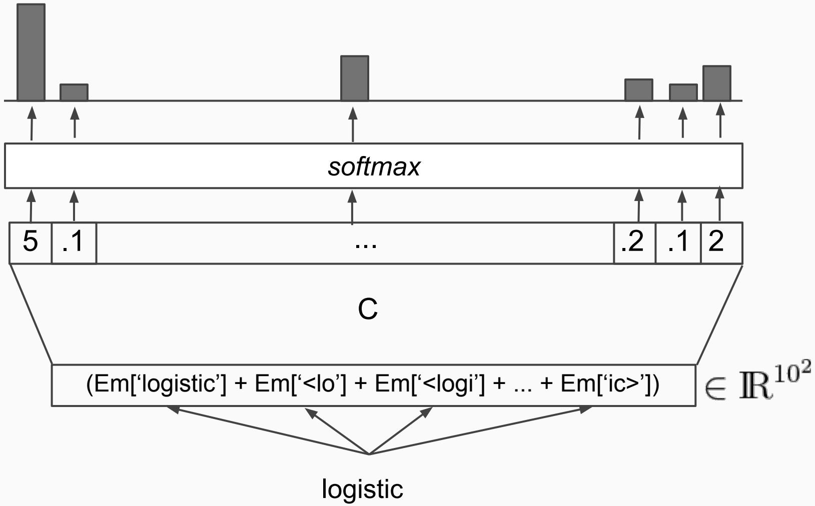
5 .1 2

C

(Em[the] + Em['black'] + Em['cat'] + … + Em['drinking milk'] + …) / 5 $\in \mathbb{R}^{10^2}$

The black cat was drinking milk from his cup .

CBoW



softmax

5 .1 ... .2 .1 2

C

(Em['logistic'] + Em['<lo'] + Em['<logi'] + ... + Em['ic>']) $\in \mathbb{R}^{10^2}$

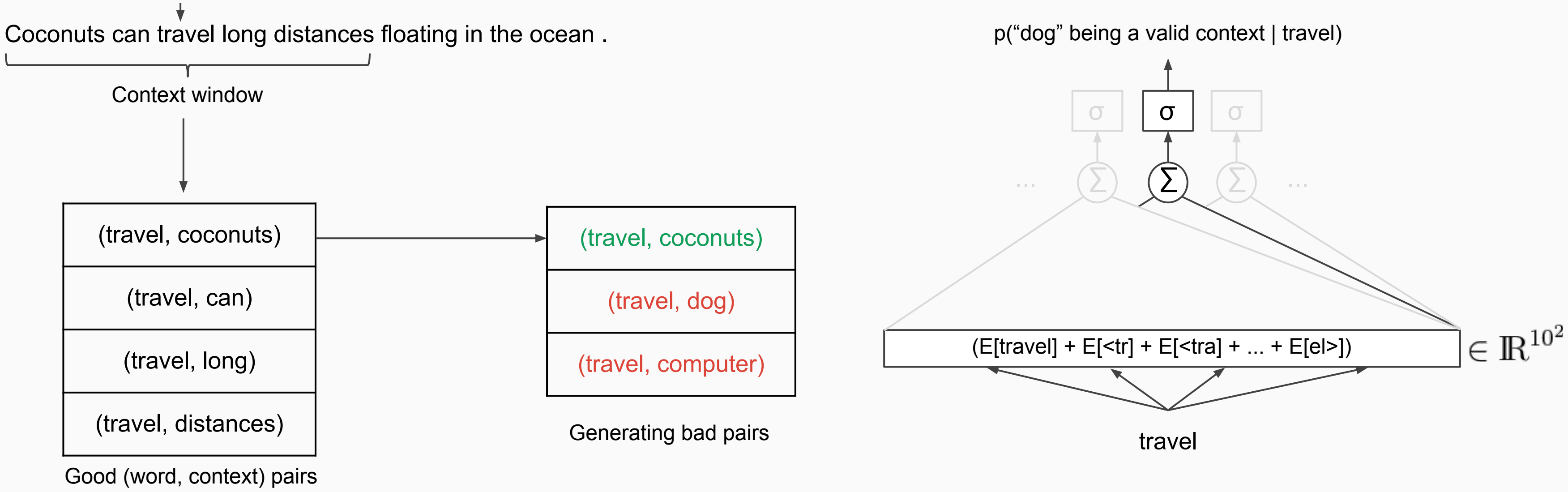
logistic

Skip-gram

Scaling up

- We saw how the supervised implementation of fasttext uses hierarchical softmax to efficiently compute the softmax loss.

- The unsupervised implementation uses NCE, like the original word2vec paper.



Coconuts can travel long distances floating in the ocean .

Context window

(travel, coconuts)

(travel, can)

(travel, long)

(travel, distances)

Good (word, context) pairs

(travel, coconuts)

(travel, dog)

(travel, computer)

Generating bad pairs

p(“dog” being a valid context | travel)

σ

Σ

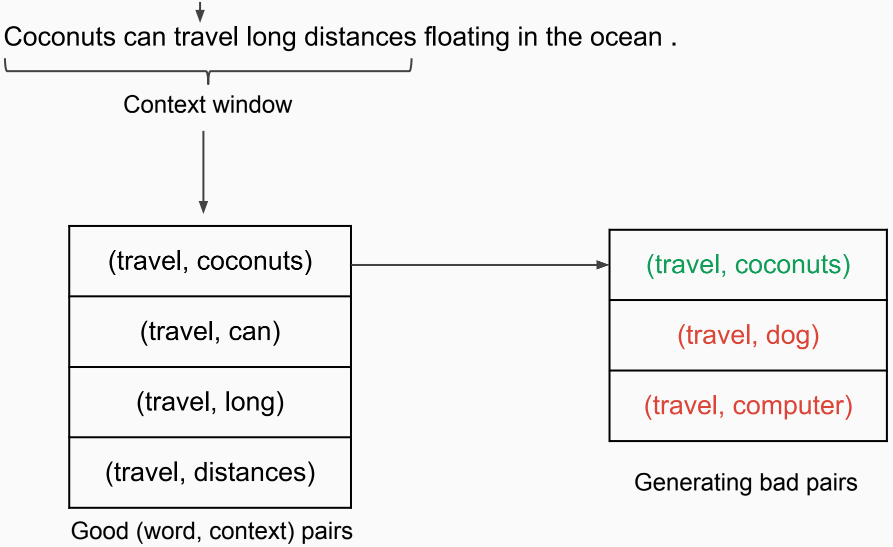
...

(E[travel] + E[<tr] + E[<tra] + ... + E[el>]) $\in \mathbb{R}^{10^2}$

travel

Scaling up

- We saw how the supervised implementation of fasttext uses hierarchical softmax to efficiently compute the softmax loss.
- The unsupervised implementation uses NCE, like the original word2vec paper.



Coconuts can travel long distances floating in the ocean .

Context window

(travel, coconuts)

(travel, can)

(travel, long)

(travel, distances)

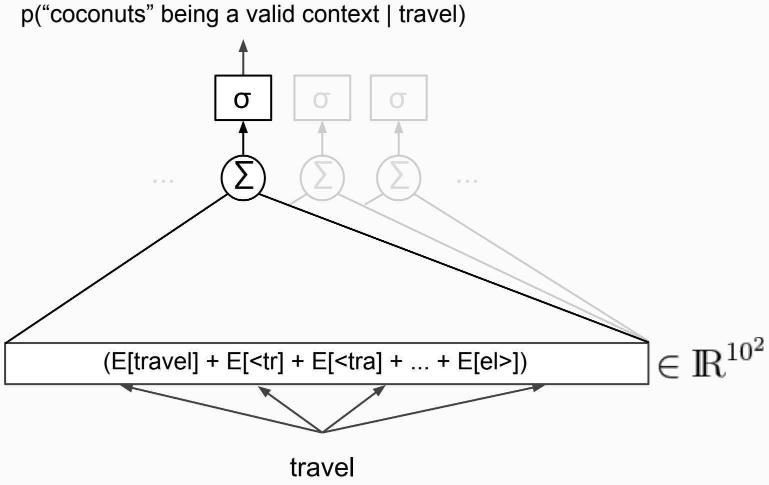
Good (word, context) pairs

(travel, coconuts)

(travel, dog)

(travel, computer)

Generating bad pairs



p(“coconuts” being a valid context | travel)

(E[travel] + E[<tr] + E[<tra] + ... + E[el>]) $\in \mathbb{R}^{10^2}$

travel

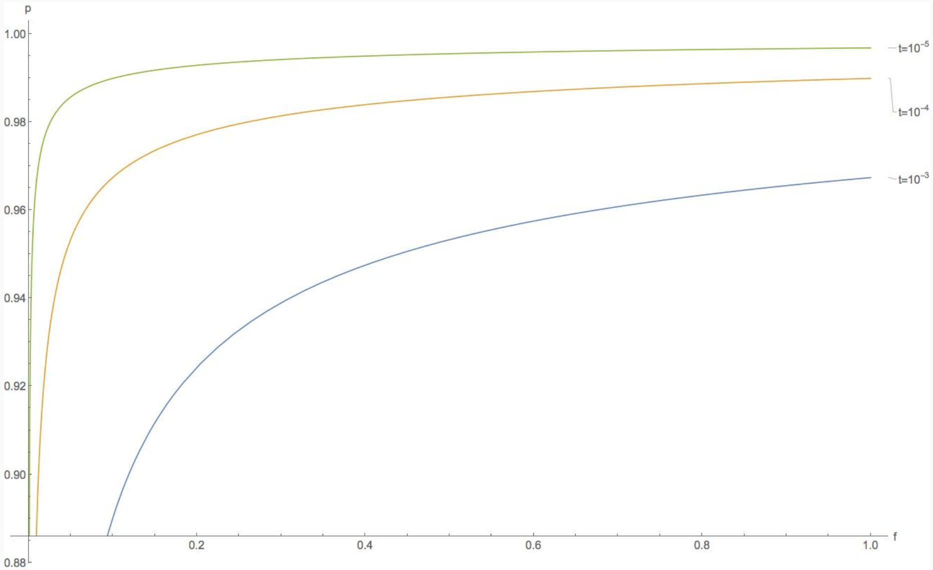
Subsampling

- Some very frequent words could behave like giant springs collapsing contexts together.
- Just as the standard word2vec, Fasttext uses subsampling:

Words are discarded with probability p:

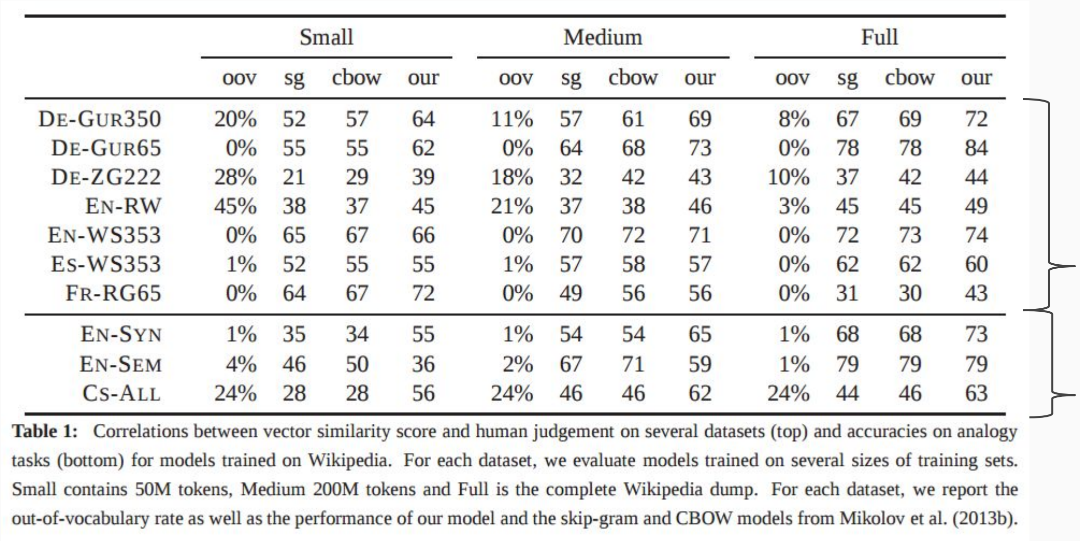
$$p = max(0, 1 - \sqrt{\frac{t}{f}} - \frac{t}{f})$$

Where f is the frequency in the corpus and
t is the subsampling threshold.



Coconuts can travel long distances floating in the ocean . → Coconuts travel distances floating ocean

Paper’s experiments



	Small				Medium				Full			
	oov	sg	cbow	our	oov	sg	cbow	our	oov	sg	cbow	our
DE-GUR350	20%	52	57	64	11%	57	61	69	8%	67	69	72
DE-GUR65	0%	55	55	62	0%	64	68	73	0%	78	78	84
DE-ZG222	28%	21	29	39	18%	32	42	43	10%	37	42	44
EN-RW	45%	38	37	45	21%	37	38	46	3%	45	45	49
EN-WS353	0%	65	67	66	0%	70	72	71	0%	72	73	74
ES-WS353	1%	52	55	55	1%	57	58	57	0%	62	62	60
FR-RG65	0%	64	67	72	0%	49	56	56	0%	31	30	43
EN-SYN	1%	35	34	55	1%	54	54	65	1%	68	68	73
EN-SEM	4%	46	50	36	2%	67	71	59	1%	79	79	79
CS-ALL	24%	28	28	56	24%	46	46	62	24%	44	46	63

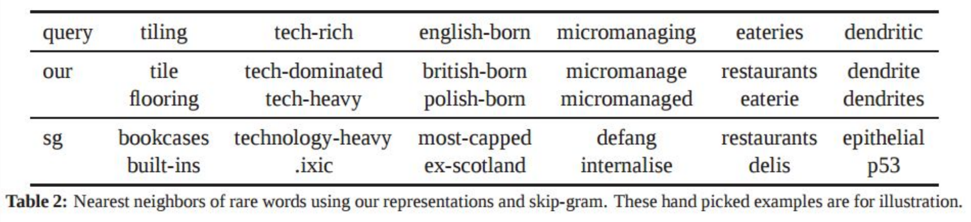
Table 1: Correlations between vector similarity score and human judgement on several datasets (top) and accuracies on analogy tasks (bottom) for models trained on Wikipedia. For each dataset, we evaluate models trained on several sizes of training sets. Small contains 50M tokens, Medium 200M tokens and Full is the complete Wikipedia dump. For each dataset, we report the out-of-vocabulary rate as well as the performance of our model and the skip-gram and CBOW models from Mikolov et al. (2013b).

Word similarity

Word analogy

Strong gain for small corpora and morphologically rich languages. Using morphemes increased overlaps

Better embeddings for rare words



query	tiling	tech-rich	english-born	micromanaging	eateries	dendritic
our	tile flooring	tech-dominated tech-heavy	british-born polish-born	micromanage micromanaged	restaurants eaterie	dendrite dendrites
sg	bookcases built-ins	technology-heavy .ixic	most-capped ex-scotland	defang internalise	restaurants delis	epithelial p53

Table 2: Nearest neighbors of rare words using our representations and skip-gram. These hand picked examples are for illustration.

Coding fasttext for word vectors

Thank you!

Questions ?

Iprova

INVENT FIRST