Simple deep learning for text classification

Friday, June 15, 2018 9:57 AM

nn for words

Neural networks for text

What is text?

You can think of text as a sequence of

- Characters
- **Words**
- Phrases and named entities
- Sentences
- Paragraphs
- …

Bag of words way (sparse)

~100k columns

	good	movie	very	a	did	like
very	0	0	**1**	0	0	0
good	**1**	0	0	0	0	0
movie	0	**1**	0	0	0	0

Bag of words way (sparse)

~100k columns

	good	movie	very	a	did	like
very	0	0	**1**	0	0	0
+						
good	**1**	0	0	0	0	0
+						
movie	0	**1**	0	0	0	0
=						
very good movie	**1**	**1**	**1**	0	0	0

Bag of words representation
is a sum of sparse one-hot-encoded vectors

Neural way (dense)

Dense representation

~300 columns

	x_1	x_2	x_3
very	**0.7**	**0.4**	**0.5**
good	**0.2**	**-0.1**	**0.1**
movie	**-0.5**	**0.4**	**0.1**

Example: word2vec embeddings

Word2vec property:
Words that have similar context tend to have collinear vectors

Neural way (dense)

~300 columns

	x_1	x_2	x_3
very	**0.7**	**0.4**	**0.5**
+			
good	**0.2**	**-0.1**	**0.1**
+			
movie	**-0.5**	**0.4**	**0.1**
=			
very good movie	**0.4**	**0.7**	**0.7**

Example: word2vec embeddings

Sum of word2vec vectors can be a good text descriptor already!

sum of w2v vector.

great baseline features for your NN models

A better way: 1D convolutions

Word embeddings

cat	**0.7**	**0.4**	**0.5**
sitting	**0.2**	**-0.1**	**0.1**
there	**-0.5**	**0.4**	**0.1**

dog	**0.6**	**0.3**	**0.5**
resting	**0.3**	**-0.1**	**0.2**
here	**-0.5**	**0.4**	**0.1**

How do we make 2-grams?

Don't have w2v embedding for n grams.

What to do? think of it as a sliding window

use a convolution filter.

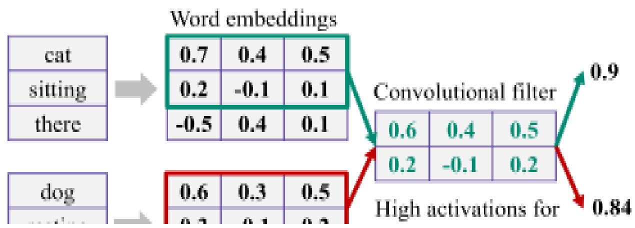
http://bionlp-www.utu.fi/wv_demo/

two gram

similar 2-grams – similar level of activation

for BOW? Two columns.

A better way: 1D convolutions



l

similar words. — similar in terms of cosine distance
(dot product)

cat sitting
≈ dog resting ⟹ So when convolving with the same filter
— similar activation
⊂ "animal sitting" Learn meaningful features

1D convolutions

- Can be extended to 3-grams, 4-grams, etc. → the advantage: dimension will not explode.
- One filter is not enough, need to track many n-grams
- They are called 1D because we slide the window only in one direction

cat	**0.7**	**0.4**	**0.5**	0.1
sitting	**0.2**	**-0.1**	**0.1**	0.3
there	**-0.5**	**0.4**	**0.1**	
or	**-0.1**	**0.8**	**-0.3**	
here	**-0.5**	**0.3**	**0.2**	

Can we take these numbers on the right as features for classification with linear model?

cat	**0.7**	**0.4**	**0.5**	0.1
sitting	**0.2**	**-0.1**	**0.1**	0.3
there	**-0.5**	**0.4**	**0.1**	-0.2
or	**-0.1**	**0.8**	**-0.3**	**0.7**
here	**-0.5**	**0.3**	**0.2**	-0.4

What to do with this vector?

- Yes
- No

Bad: output = number of input.

If we can afford the order of the text (assumption)

=> Take the max. of the conv.

Max Pooling Over time

1D convolutions

- Can be extended to 3-grams, 4-grams, etc.
- One filter is not enough, need to track many n-grams
- They are called 1D because we slide the window only in one direction

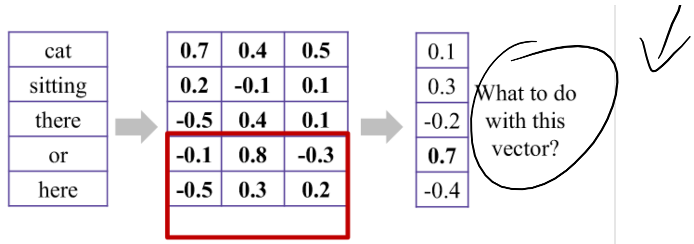
cat	**0.7**	**0.4**	**0.5**	0.1
sitting	**0.2**	**-0.1**	**0.1**	0.3
there	**-0.5**	**0.4**	**0.1**	-0.2
or	**-0.1**	**0.8**	**-0.3**	0.7
here	**-0.5**	**0.3**	**0.2**	

1D convolutions

- Can be extended to 3-grams, 4-grams, etc.
- One filter is not enough, need to track many n-grams
- They are called 1D because we slide the window only in one direction

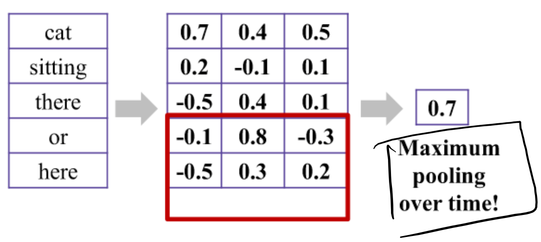
cat	**0.7**	**0.4**	**0.5**	0.1
sitting	**0.2**	**-0.1**	**0.1**	0.3

What to do



1D convolutions

- Can be extended to 3-grams, 4-grams, etc.
- One filter is not enough, need to track many n-grams
- They are called 1D because we slide the window only in one direction



Let’s train many filters

Final architecture

- 3,4,5-gram windows with $\boxed{100}$ filters each
- MLP on top of these 300 features

Quality comparison on customer reviews (CR)

- Naïve Bayes on top of 1,2-grams – 86.3% accuracy
- 1D convolutions with MLP – 89.6% (+3.8%) accuracy

http://www.jmlr.org/papers/volume12/collobert11a/collobert11a.pdf https://arxiv.org/pdf/1408.5882.pdf

experiment papers

Compare with BoW model.

Summary

- You can just average pre-trained word2vec vectors for your text
- You can do better with 1D convolutions that learn more complex features
- In the next video we'll continue to apply convolutions to text

nn for character

Going deeper with text

What is text?

You can think of text as a sequence of

- **Characters**
- Words
- Phrases and named entities
- Sentences
- Paragraphs
- …

Text as a sequence of characters

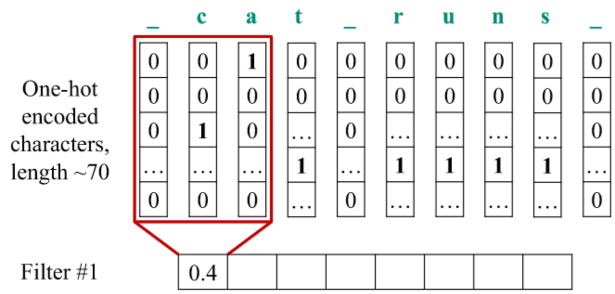
One-hot encoded characters, length ~70

_	c	a	t	_	r	u	n	s	_
0	0	**1**	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	**1**	0	…	0	…	…	…	…	0
…	…	…	**1**	…	**1**	**1**	**1**	**1**	…
0	0	0	…	0	…	…	…	…	0

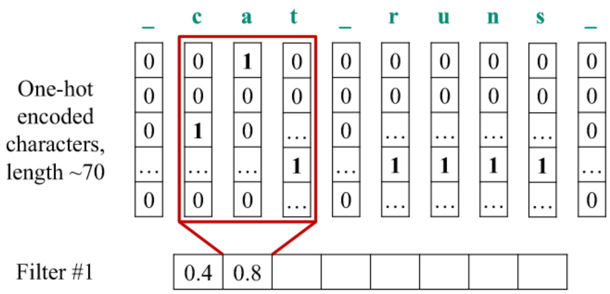
will be sparse, but not long.

Let's start with character *n*-grams!

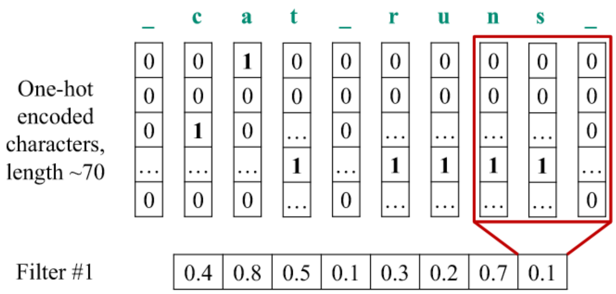
1D convolutions on characters



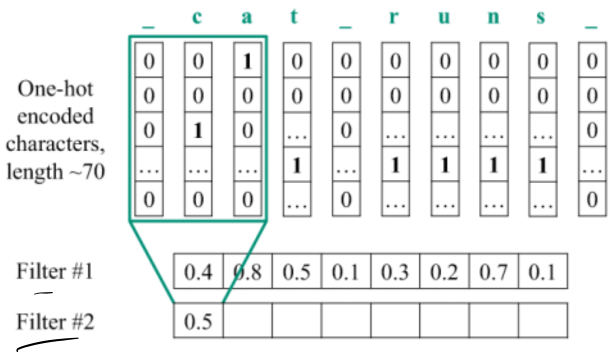
1D convolutions on characters



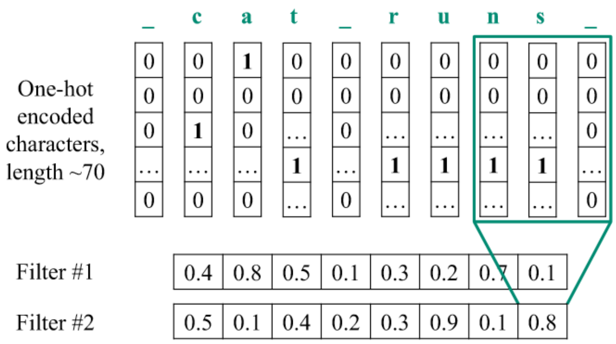
1D convolutions on characters



1D convolutions on characters



1D convolutions on characters



1D convolutions on characters

One-hot encoded characters, length ~70

_	c	a	t	_	r	u	n	s	_
0	0	**1**	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	**1**	0	…	0	…	…	…	…	0
…	…	…	**1**	…	**1**	**1**	**1**	**1**	…
0	0	0	…	0	…	…	…	…	0

Filter #1	0.4	0.8	0.5	0.1	0.3	0.2	0.7	0.1
Filter #2	0.5	0.1	0.4	0.2	0.3	0.9	0.1	0.8
Filter #3	0.4	0.7	0.3	0.7	0.5	0.5	0.9	0.4

~1024 filters

What's next? Let's add pooling!

Max pooling

Filter #1	0.4	0.8	0.5	0.1	0.3	0.2	0.7	0.1
Filter #2	0.5	0.1	0.4	0.2	0.3	0.9	0.1	0.8
Filter #3	0.4	0.7	0.3	0.7	0.5	0.5	0.9	0.4

0.8			

Max pooling

Filter #1	0.4	0.8	0.5	0.1	0.3	0.2	0.7	0.1
Filter #2	0.5	0.1	0.4	0.2	0.3	0.9	0.1	0.8
Filter #3	0.4	0.7	0.3	0.7	0.5	0.5	0.9	0.4

0.8	0.5		

Max pooling

Filter #1	0.4	0.8	0.5	0.1	0.3	0.2	0.7	0.1
Filter #2	0.5	0.1	0.4	0.2	0.3	0.9	0.1	0.8
Filter #3	0.4	0.7	0.3	0.7	0.5	0.5	0.9	0.4

0.8	0.5	0.3	0.7

Max pooling

Filter #1	0.4	0.8	0.5	0.1	0.3	0.2	0.7	0.1
Filter #2	0.5	0.1	0.4	0.2	0.3	0.9	0.1	0.8
Filter #3	0.4	0.7	0.3	0.7	0.5	0.5	0.9	0.4

Pooling output

0.8	0.5	0.3	0.7
0.5	0.4	0.9	0.8
0.7	0.7	0.5	0.9

Provides a little bit of position invariance for character n-grams → moving n-gram to the left or right

⇒ after pooling the output stay the same

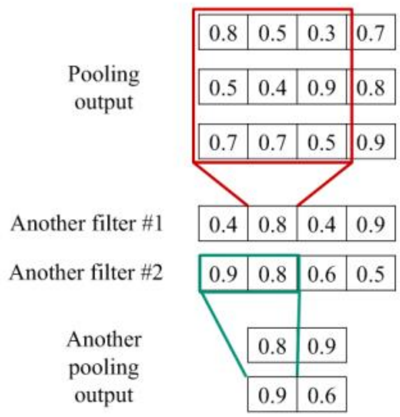
Repeat 1D convolution + pooling

Pooling output

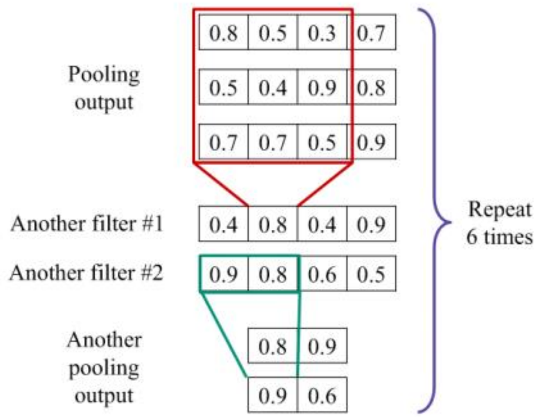
0.8	0.5	0.3	0.7
0.5	0.4	0.9	0.8
0.7	0.7	0.5	0.9

Another filter #1	0.4	0.8	0.4	0.9
Another filter #2	0.9	0.8	0.6	0.5

Repeat 1D convolution + pooling



Repeat 1D convolution + pooling



0.6
0.6
0.3
0.2

Final architecture

- Let's take only first **1014** characters of text
- Apply 1D convolution + max pooling **6** times
  - Kernels widths: 7, 7, 3, 3, 3, 3
  - Filters at each step: 1024
- After that we have a $\mathbf{1024} \times \mathbf{34}$ matrix of features
- Apply MLP for your task

→ can be regression classification
or other cost you like.

https://arxiv.org/pdf/1509.01626.pdf

Experimental datasets

Categorization or sentiment analysis

	Dataset	Classes	Train Samples
Smaller	AG's News	4	120,000
	Sogou News	5	450,000
	DBPedia	14	560,000
	Yelp Review Polarity	2	560,000
Bigger	Yelp Review Full	5	650,000
	Yahoo! Answers	10	1,400,000
	Amazon Review Full	5	3,000,000
	Amazon Review Polarity	2	3,600,000

https://arxiv.org/pdf/1509.01626.pdf

Experimental results

Errors on test set for classical models:

Model	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
BoW	11.19	7.15	3.39	7.76	42.01	31.11	45.36	9.60
BoW TFIDF	10.36	6.55	2.63	6.34	40.14	28.96	44.74	9.00
ngrams	7.96	2.92	1.37	**4.36**	43.74	31.53	45.73	7.98
ngrams TFIDF	**7.64**	**2.81**	**1.31**	4.56	45.20	31.49	47.56	8.46

Errors on test set for deep models:

Model	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
LSTM	13.94	4.82	1.45	5.26	41.83	29.16	40.57	6.10
Sm. Full Conv.	11.59	8.95	1.89	5.67	38.82	30.01	40.88	5.78
Lg. Full Conv. Th.	9.51	-	1.55	4.88	38.04	29.58	40.54	5.51
Sm. Full Conv. Th.	10.89	-	1.69	5.42	**37.95**	29.90	40.53	5.66
Lg. Conv.	12.82	4.88	1.73	5.89	39.62	29.55	41.31	5.51
Sm. Conv.	15.65	8.65	1.98	6.53	40.84	29.84	40.53	5.50
Lg. Conv. Th.	13.39	-	1.60	5.82	39.30	**28.80**	40.45	**4.93**
Sm. Conv. Th.	14.80	-	1.85	6.49	40.16	29.84	**40.43**	5.67

Deep models work better for large datasets!

https://arxiv.org/pdf/1509.01626.pdf

Summary

- You can use convolutional networks on top of characters (called learning from scratch)
- It works best for large datasets where it beats classical approaches (like BOW)
- Sometimes it even beats LSTM that works on word level

1. Let's recall how we treated words as one-hot sparse vectors in BOW and dense embeddings in neural networks:

~100k columns

	good	movie	very	a	did	like
very	0	0	1	0	0	0

~300 columns

	x_1	x_2	x_3
very	0.7	0.4	0.5

Choose correct statements below.

- [x] For **both** word representations we can take a **sum** of vectors corresponding to tokens of any text to obtain good features for this text for further usage in linear model.

Correct
Yes, this is true. Don't forget to normalize these features row-wise!

- [ ] You can replace **word2vec** embeddings with any **random** vectors to get a good features descriptor as a **sum** of vectors corresponding to all text tokens.

Un-selected is correct

- [x] Linear model on top of a **sum** of neural representations can work faster than on top of BOW.

Correct
This is true! We only need to train 300 parameters here. Don't forget to normalize these features row-wise!

- [x] For **both** word representations we can take a **weighted sum** of vectors corresponding to tokens of any text to obtain good features for this text for further usage in linear model. The **weight** for any token can be an IDF value for that token.

Correct
Yes, this is true. For BOW we effectively get bag of TF-IDF values, where TF is a binary variable. Don't forget to normalize these features row-wise!