Function space analysis of NLP models

Uri Katz

Danielle Haulser

Intro

- Text representation are very different from image representation.

- As opposed to image representation which deals with continuous structures,
- Text representations are built from discrete symbol units (e.g. letter, words)

- Our goal was to use function space theory to better understand and quantify the geometry of NLP models.

- We have measured the smoothness and accuracy of layers of text classification CNN and compared between different methods of vector representations of words.

NLP models – Natural language processing

- NLP is the field that unite all methods that combines linguistics and computer interactions
- Machine translation , Search engines , Chatbots , Text classification are all common applications of NLP
- Similarly to Computer vision , Top competitors and contributors in the field are the big tech companies such as Google , Facebook and also Stanford NLP lab.

NLP models – Embedding naive methods

- Mapping words/ phrases from the vocabulary into vectors of real numbers

- One hot encoding :

Each location in the vector represents a different colour

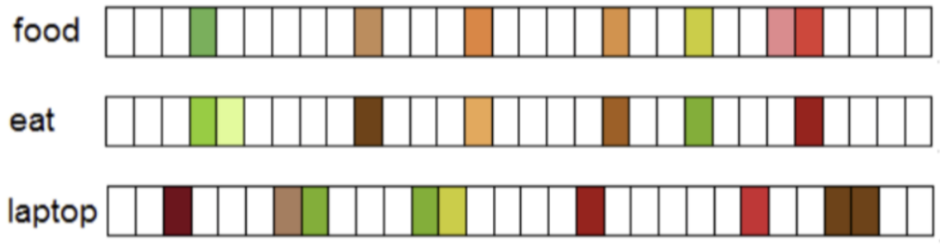
```
orange = [1,0,0,0,0,0,0,0...]
```

- Dimensions are very high
- Features are completely independent from one another
- occurrences of ‘dog’ will not tell us anything about the occurrences of ‘cat’

NLP models – Embedding methods

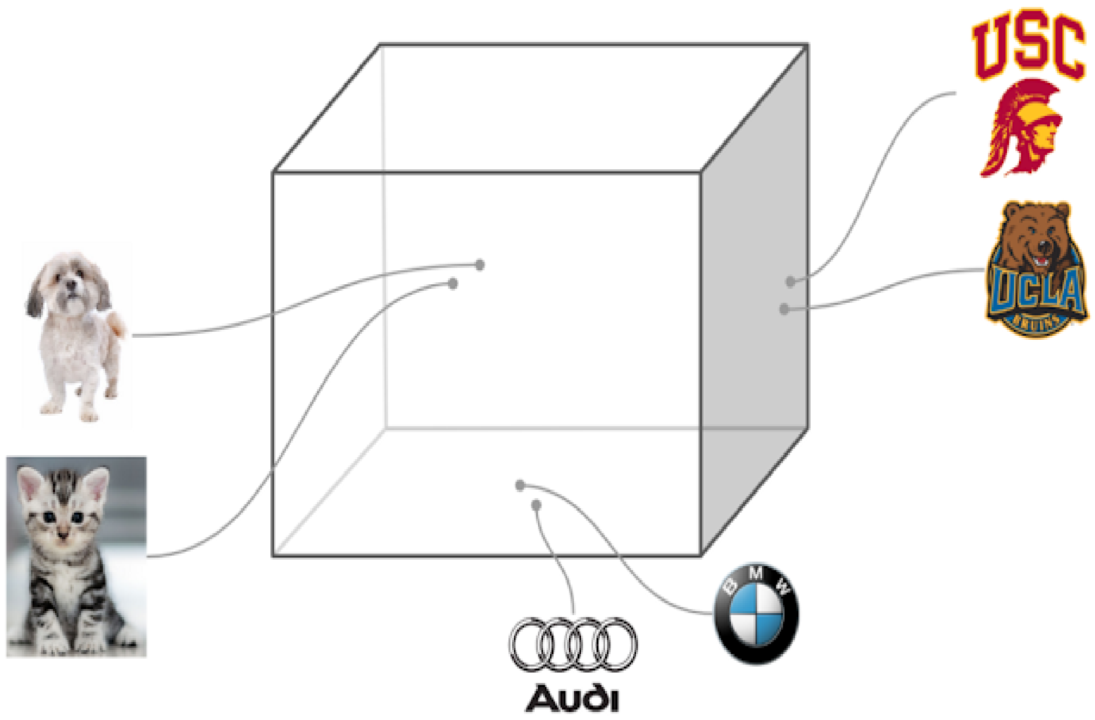
Dense representations :

- Dimensionality of vector is d
- Similar features will have similar vectors – information is shared between similar features
- Example : distributed word vectors
  - Words are represented as a distribution of its membership to each of the features.



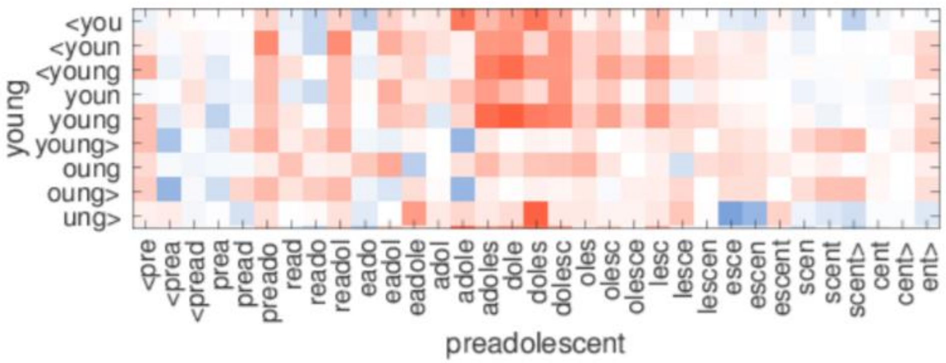
Distributed word vectors

NLP models – Encoding words based on similarity



- Embed words based on their relationship and similarity.
- Word are being transformed into real vectors taken from high dimensional continuous space.
- These vectors holds relations that can be quantify by mean of similarity (the cosine of there angle) and can be used for clustering purposes as well,

FastText pretrained vectors



EX) where / n = 3, it will be represented by the character n-grams :

<wh / whe / $\boxed{\text{her}}$ / ere / re>

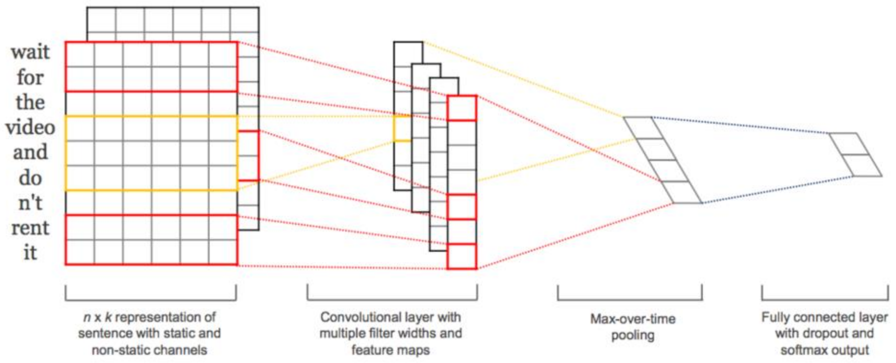
And the special sequence

<where>

word *<her>* is different from the tri-gram *her* from the word where *where*

- Standard word vectors ignore word internal structure
- useful information for rare or misspelled words
- enriched word vectors with a bag of character n-gram vectors
- derived from a large corpus of data

CNN for text classification



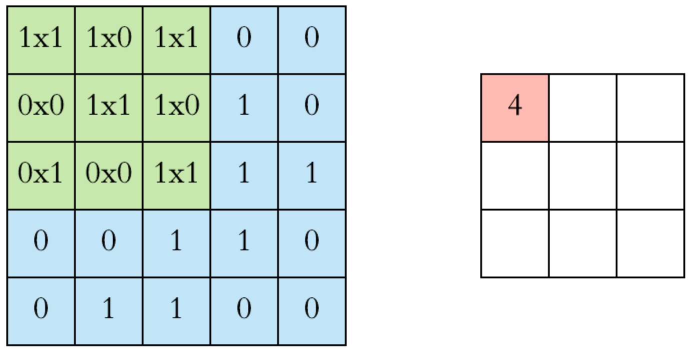
- CNN is designed to identify local predictors in a large structure
- produce a fixed size vector representation of the structure
- Captures the local aspects that are most informative for the prediction task .

Function space representation – CNN layers

- A dataset of sentences of dimension $d_{\text{(word vec length)}} \times n_{\text{(sentence length)}}$
- Each word is embedded to a vector of size 300 , each sentence is 300 × 20
- The vector values are normalized to [-1,1]
- Each sentence is labeled as : “positive” or “negative”
- For layer 0 :each sentence is a sample of a function f_0: $[-1,1]^{d \times n}$-> $\mathbb{R}^1$

- Similar process for the inner layers
- For each K-th layer, will have samples of a function :
- f_k: $[-1,1]^{d_k \times n_k}$-> $\mathbb{R}^1$

CNN for text classification – : 1D Text Convolutions



- m filters
- n-words input text: $w_1, \ldots, w_n \in R^{\mathrm{d}}$ embedded as a d dimensional vector
- The d × n matrix is fed into a convolutional layer
- we pass a sliding window over the text. For each ℓ-words ngram:
- $\boldsymbol{u_i} = [w_i, \ldots, w_{i+\ell-1}] \in R^{\mathrm{d}\times\ell}$; $0 \leq i \leq n-\ell$
- for each filter $\boldsymbol{f_j} \in R^{\mathrm{d}\times\ell}$ we calculate $F_{ij} = \langle \boldsymbol{u}_{\mathrm{i}}, \boldsymbol{f_j} \rangle$, $F \in R^{\mathrm{n}\times m}$
- Usually applying max-pooling across the ngram dimension

CNN for text classification

- filters serve as ngram detectors
- each filter searches for a specific class of ngrams, and assigning them high scores.
- The highest-scoring detected ngrams survive the max-pooling operation.
- The final decision is then based on the set of ngrams in the max-pooled vector

Smoothness analysis -

- How can we quantify the geometry of the clustering within each layer representation?

$$|f|_{B_{\mathcal{T}}^{\alpha,r}(F)} := \frac{1}{J}\left(\sum_{j=1}^{J} |f|_{B_{\mathcal{T}}^{\alpha,r}(\mathcal{T}_j)}\right)^{1/\mathcal{T}}$$

$$\frac{1}{\mathcal{T}} = \alpha + \frac{1}{p}$$

- The Besov index of f is determined by the maximal index α.
- The higher the index α, the smoother the function is.

Smoothness analysis -

- Jackson theorem , (for r=1):

$$\sigma_m \coloneqq \|f - \mathcal{F}_M\|_P \leq (C_P, \alpha, P)JM \leq |f|{B_\tau}^{\alpha,1}(F)$$

With the discrete error of the wavelet M-term approximation (p=2)

$$\sigma_M(f)^2 = \frac{1}{|m|} \sum_{1i=1}^{m} \|\mathcal{F}_M(x_i) - f(x_i)\|_{\ell_2(L-1)}.$$

- so, we can model the error function by:

$$\sigma_m \sim CM^{-\alpha}$$

$$\log\big(\sigma_m(f)\big) \sim \log C - \alpha \log M$$

- for α , C we can solve through least squares

- In the context of DL the Besov index of smoothness can be applied on the training set in the feature space of each layer.

Applications

- Sentiment140 dataset - 1,600,000 tweets.
- CNN models created based on TensorFlow (Keras) networks models.
- The train executed on a GPU computer with 30 GB of RAM.
- The smoothness analysis executed on Microsoft Azure cloud computing platform with a DS14 virtual machine - 16 CPUs and 140 GB RAM.

Dataset : Sentiment140

- 1,600,000 tweets annotated for positive and negative sentiment :

Negative :

"uh oh, Dr. Phil just made me cry and it looks like Oprah is going to do the same "

Positive :

"taylor swift's songs make me happy, i don't know why haha "

Dataset : Sentiment140

- 12% of the words in the dataset are unrecognizable due to the use of slang / hashtags / named entities . i.e :

yumm , cyrus , aweesome, wuvvv , woot , f0r , covfefe, hoooot

- Unrecognizable words get random weights at the beginning of the training .

Project’s CNN architectures – One hot encoding layer

Layer	Input Shape	Output Shape	Param [#]	Activation Type
One hot	(None, 20, 300)	(None, 20, 300)	634200	-
Conv1D	(None, 20, 300)	(None, 18, 600)	540600	'relu'
Flatten	(None, 18, 600)	(None, 10800)	0	-
Dense	(None, 10800)	(None, 1)	10801	-
Activation	(None, 1)	(None, 1)	0	'sigmoid'

Project’s CNN architectures – Embedding layer

Layer	Input Shape	Output Shape	Param [#]	Activation Type
Embedding	(None, 20, 300)	(None, 20, 300)	634200	-
Conv1D	(None, 20, 300)	(None, 18, 600)	540600	'relu'
Flatten	(None, 18, 600)	(None, 10800)	0	-
Dense	(None, 10800)	(None, 1)	10801	-
Activation	(None, 1)	(None, 1)	0	'sigmoid'

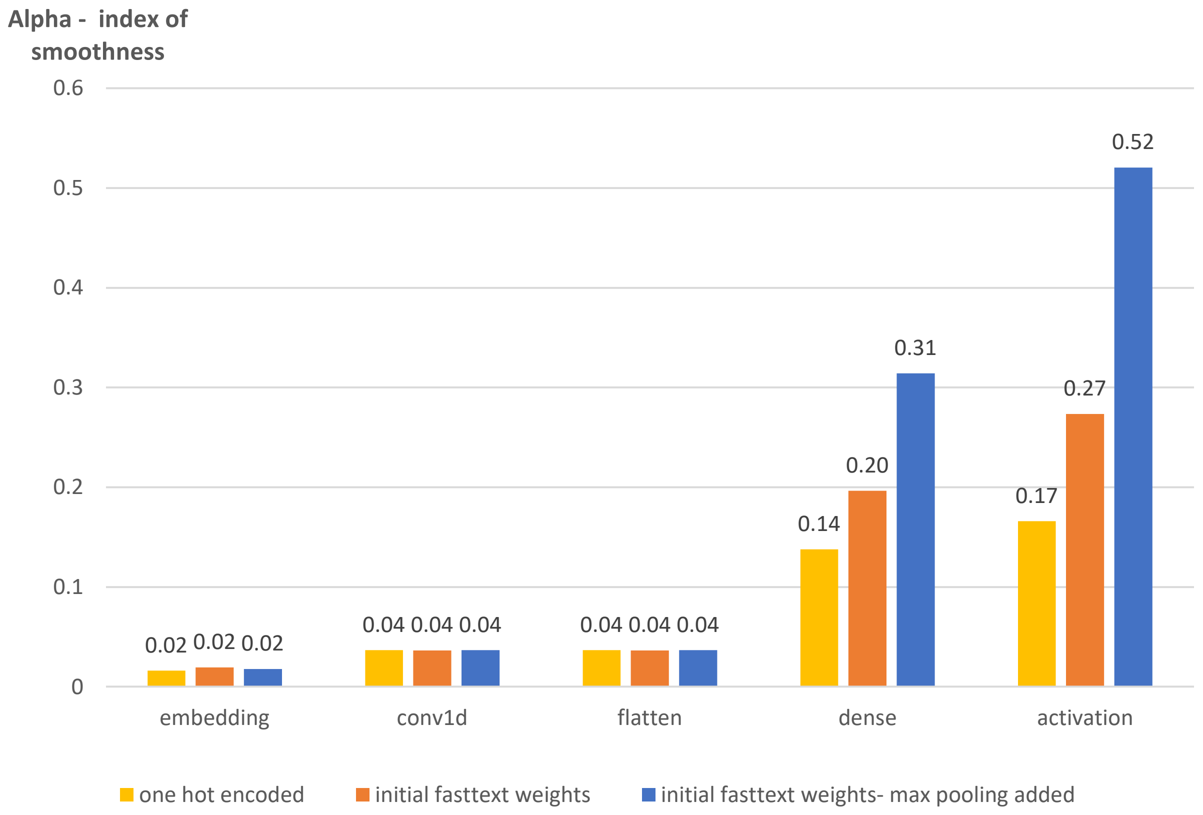
Project's CNN architectures – Embedding layer, and max pooling added

Layer	Input Shape	Output Shape	Param [#]	Activation Type
Embedding	(None, 20, 300)	(None, 20, 300)	634200	-
Conv1D	(None, 20, 300)	(None, 18, 600)	540600	'relu'
MaxPooling1D	(None, 18, 600)	(None, 9, 600)	0	-
Flatten	(None, 9, 600)	(None, 5400)	0	-
Dense	(None, 5400)	(None, 1)	5401	-
Activation	(None, 1)	(None, 1)	0	'sigmoid'

Results

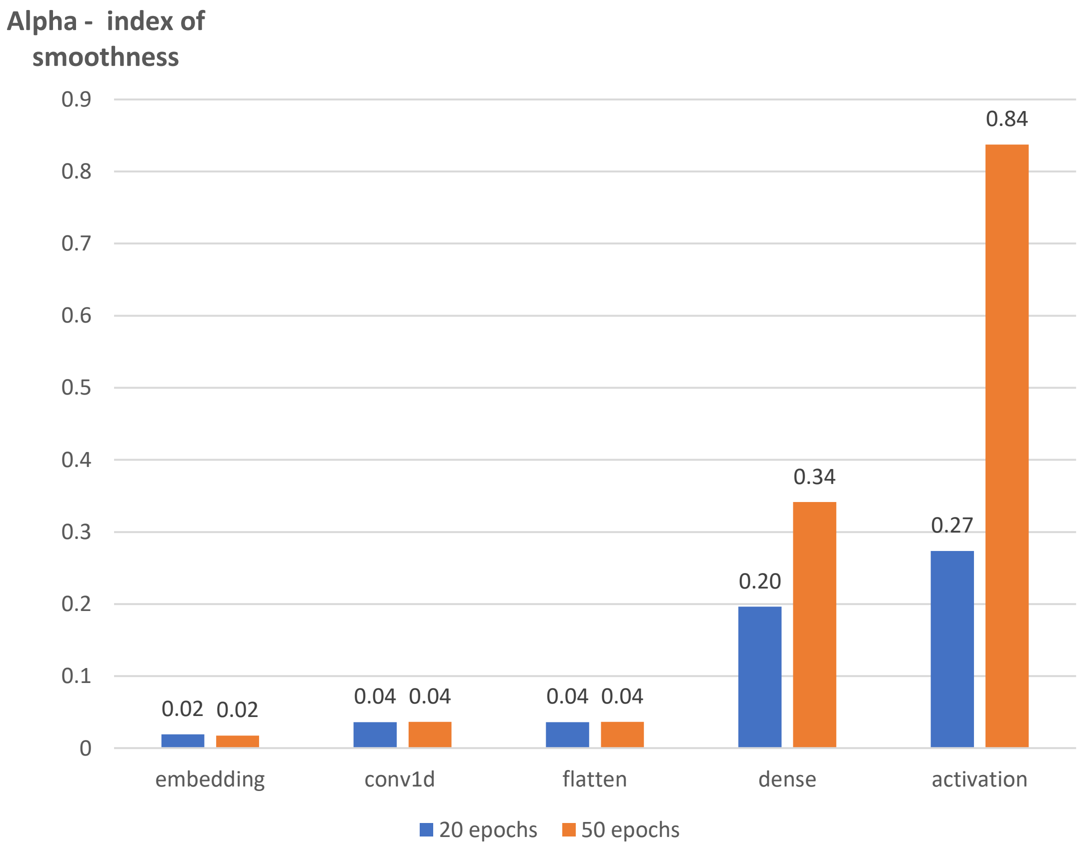
Model	Epochs	Hyper parameters	Accuracy : (TP+TN)/(P+N)	Precision : TP/(TP+FP)	Recall: TP/(TP+FN)
1. CNN, encoded with one hot method.	20	embed dim = 300 filters = 300 kernel size = 3	0.65 ± 0.0008	0.65 ± 0.0032	0.654 ± 0.0015
2. CNN, Embedded Based on fasttext pre trained vectors	20	embed dim = 300 filters = 300 kernel size = 3	0.742 ± 0.0014	0.738 ± 0.006	0.744 ± 0.01
3. CNN, Embedded Based on fasttext pre trained vectors	50	embed dim = 300 filters = 300 kernel size = 3	0.742 ± 0.0015	0.741 ± 0.004	0.744 ± 0.0081
4. CNN, Embedded Based on fasttext pre trained vectors, with max pooling layer added	20	embed dim = 300 filters = 300 kernel size = 3	0.743 ± 0.0008	0.743 ± 0.0033	0.745 ± 0.0092
5. CNN, Embedded Based on fasttext pre trained vectors	20	embed dim = 300 filters = 300 kernel size = 5	0.748 ± 0.001	0.746 ± 0.0035	0.748 ± 0.0073

Smoothness analysis s of DL layers representations–



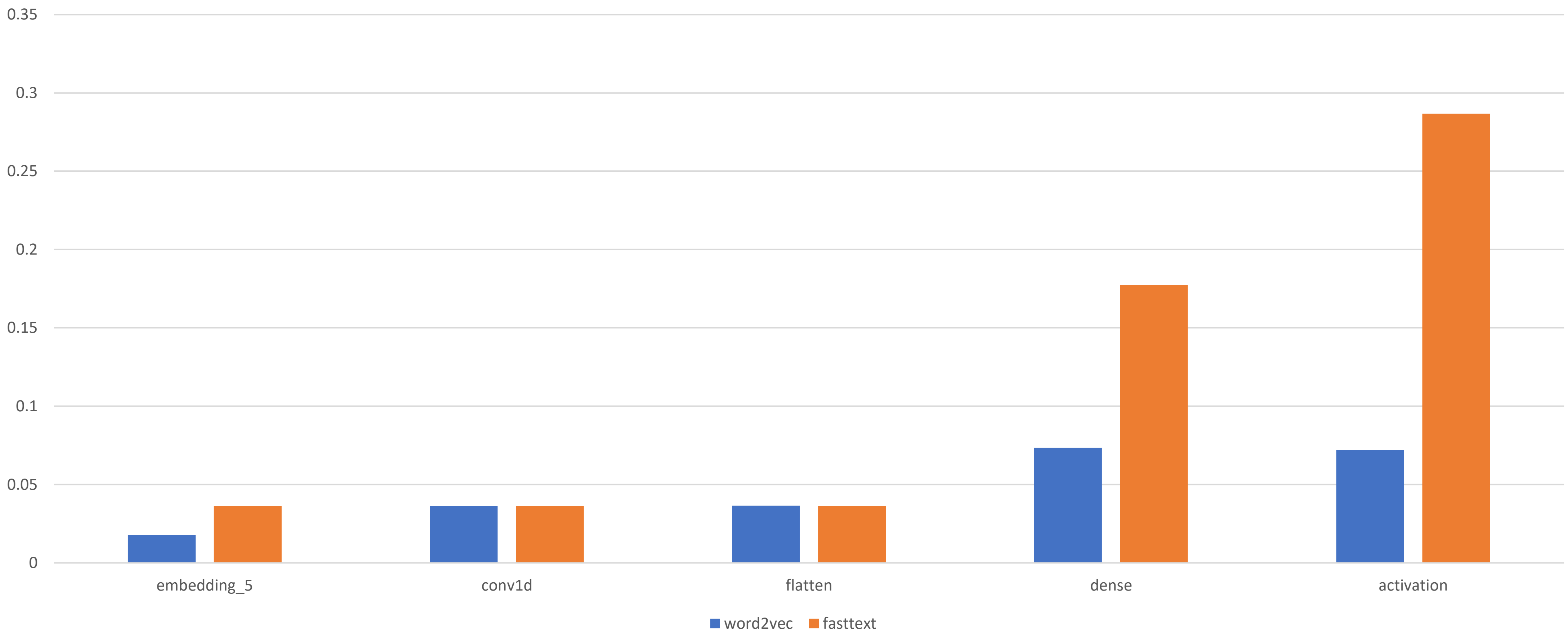
- The Besov α-index increases from layer to layer - the clustering improves
- The smoothness of the last layers where we use initial weights instead of one hot encoding is higher - importance to the initial representation of the data.
- At the last layers, the smoothness where added max pooling (after convolution layer) is the highest.
- Complies with the theory, Max-pooling extracts the relevant ngrams for making a decision.

Smoothness analysis s of DL layers representations– 20 compare to 50 epoch



- The smoothness increased from layer to layer
- Smoothness improved where we used more epochs.

Word2vec compare to fasttext



Discussion

- We have seen that using embedding with fasttext vectors yields a better clustering – and results in a higher index of smoothness.

- That is opposed to the naïve methods that don't create geometrical relationships within the feature space.

- It might be interesting to further investigate the relationship between the feature space to the accuracy and smoothness of the data:

  - Can we choose the k- most important features and still get the same accuracy or maybe higher accuracy?
  - How will the index of smoothness change , according to that?