Nonsymbolic Text Representation

Hinrich Schütze
CIS, LMU Munich, Germany
inquiries@cislmu.org

Abstract

We introduce the first generic text representation model that is completely nonsymbolic, i.e., it does not require the availability of a segmentation or tokenization method that attempts to identify words or other symbolic units in text. This applies to training the representations as well as to using them in an application. We demonstrate better performance than prior work on entity typing and text denoising.

1 Introduction

Character-level models can be grouped into three classes. (i) **End-to-end models** learn a separate model on the raw character (or byte) input for each task; these models estimate task-specific parameters, but no representation of text that would be usable across tasks is computed. Throughout this paper, we refer to $r(x)$ as the "representation" of x only if $r(x)$ is a generic rendering of x that can be used in a general way, e.g., across tasks and domains. The activation pattern of a hidden layer for a given input sentence in a multilayer perceptron (MLP) is not a representation according to this definition if it is not used outside of the MLP. (ii) **Character-level models of words** derive a representation of a word w from the character string of w, but they are symbolic in that they need text segmented into tokens as input. (iii) Bag-of-character-ngram models, **bag-of-ngram models** for short, use character ngrams to encode sequence-of-character information, but sequence-of-ngram information is lost in the representations they produce.

Our premise is that text representations are needed in NLP. A large body of work on word embeddings demonstrates that a generic text representation, trained in an unsupervised fashion on large corpora, is useful. Thus, we take the view that **group (i) models**, end-to-end learning without any representation learning, is not a good general approach for NLP.

We distinguish **training** and **utilization** of the text representation model. We use "training" to refer to the method by which the model is learned and "utilization" to refer to the application of the model to a piece of text to compute a representation of the text. In many text representation models, utilization is trivial. For example, for word embedding models, utilization amounts to a simple lookup of a word to get its precomputed embedding. However, for the models we consider, utilization is not trivial and we will discuss different approaches.

Both training and utilization can be either **symbolic** or **nonsymbolic**. We define a symbolic approach as one that is based on tokenization, i.e., a segmentation of the text into tokens. Symbol identifiers (i.e., tokens) can have internal structure – a tokenizer may recognize tokens like "to and fro" and "London-based" that contain delimiters – and may be morphologically analyzed downstream.[1]

We define a nonsymbolic approach as one that is tokenization-free, i.e., no assumption is made that there are segmentation boundaries and that each segment (e.g., a word) should be represented (e.g., by a word embedding) in a way that is independent of the representations (e.g., word embeddings) of neighboring segments. Methods for training text representation models that require tokenized text include word embedding models like word2vec (Mikolov et al., 2013) and most **group**

[1]The position-embedding representation of a text introduced below is a sequence of position embeddings. An embedding that represents a single character must be viewed as symbolic since a character is a symbol – just like a representation of text as a sequence of word embeddings is symbolic since each word corresponds to a symbol. But position embeddings do not represent single characters. See §4.

Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, pages 785–796,
Valencia, Spain, April 3-7, 2017.

(ii) methods, i.e., character-level models like fastText skipgram (Bojanowski et al., 2016).

Bag-of-ngram models, group (iii) models, are text representation utilization models that typically compute the representation of a text as the sum of the embeddings of all character ngrams occurring in it, e.g., WordSpace (Schütze, 1992) and CHARAGRAM (Wieting et al., 2016). WordSpace and CHARAGRAM are examples of mixed training-utilization models: training is performed on tokenized text (words and phrases), utilization is nonsymbolic.

We make two contributions in this paper. (i) We propose the first generic method for training text representation models without the need for tokenization and address the challenging sparseness issues that make this difficult. (ii) We propose the first nonsymbolic utilization method that fully represents sequence information – in contrast to utilization methods like bag-of-ngrams that discard sequence information that is not directly encoded in the character ngrams themselves.

2 Motivation

Chung et al. (2016) give two motivations for their work on character-level models. First, **tokenization** (or, equivalently, segmentation) **algorithms make many mistakes** and are brittle: "we do not have a perfect word segmentation algorithm for any one language". Tokenization errors then propagate throughout the NLP pipeline.

Second, there is currently **no general solution for morphology** in statistical NLP. For many languages, high-coverage and high-quality morphological resources are not available. Even for well resourced languages, problems like ambiguity make morphological processing difficult; e.g., "rung" is either the singular of a noun meaning "part of a ladder" or the past participle of "to ring". In many languages, e.g., in German, syncretism, a particular type of systematic morphological ambiguity, is pervasive. Thus, there is no simple morphological processing method that would produce a representation in which all inflected forms of "to ring" are marked as having a common lemma; and no such method in which an unseen form like "aromatizing" is reliably analyzed as a form of "aromatize" whereas an unseen form like "anti-trafficking" is reliably analyzed as the compound "anti+trafficking".

Of course, it is an open question whether nonsymbolic methods can perform better than morphological analysis, but the foregoing discussion motivates us to investigate them.

Chung et al. (2016) focus on problems with the tokens produced by segmentation algorithms. Equally important is the problem that **tokenization fails to capture structure across multiple tokens**. The job of dealing with cross-token structure is often given to downstream components of the pipeline, e.g., components that recognize multiwords and named entitites in English or in fact any word in a language like Chinese that uses no overt delimiters. However, there is no linguistic or computational reason in principle why we should treat the recognition of a unit like "electromechanical" (containing no space) as fundamentally different from the recognition of a unit like "electrical engineering" (containing a space). Character-level models offer the potential of uniform treatment of such linguistic units.

3 Text representation model: Training

3.1 Methodology

Many text representation learning algorithms can be understood as estimating the parameters of the model from a unit-context matrix C where each row corresponds to a unit u_i, each column to a context c_j and each cell C_{ij} measures the degree of association between u_i and c_j. For example, the skipgram model is closely related to an SVD factorization of a pointwise mutual information matrix (Levy and Goldberg, 2014). Many text representation learning algorithms are formalized as matrix factorization (e.g., (Deerwester et al., 1990; Hofmann, 1999; Stratos et al., 2015)), but there may be no big difference between implicit (e.g., (Pennington et al., 2014)) and explicit factorization methods; see also (Mohamed, 2011; Rastogi et al., 2015).

Our goal in this paper is not to develop new matrix factorization methods. Instead, we will focus on defining the unit-context matrix in such a way that no symbolic assumption has to be made. This unit-context matrix can then be processed by any existing or still to be invented algorithm.

Definition of units and contexts. How to define units and contexts without relying on segmentation boundaries? In initial experiments, we simply generated all character ngrams of length up to k_{max} (where k_{max} is a parameter), including character ngrams that cross token boundaries; i.e., no

segmentation is needed. We then used a skipgram-type objective for learning embeddings that attempts to predict, from ngram g_1, an ngram g_2 in g_1's context. Results were poor because many training instances consist of pairs (g_1, g_2) in which g_1 and g_2 overlap, e.g., one is a subsequence of the other. So the objective encourages trivial predictions of ngrams that have high string similarity with the input and nothing interesting is learned.

In this paper, we propose an alternative way of defining units and contexts that supports well-performing nonsymbolic text representation learning: **multiple random segmentation**. A pointer moves through the training corpus. The current position i of the pointer defines the left boundary of the next segment. The length l of the next move is uniformly sampled from $[k_{\text{min}}, k_{\text{max}}]$ where k_{min} and k_{max} are the minimum and maximum segment lengths. The right boundary of the segment is then $i+l$. Thus, the segment just generated is $c_{i,i+l}$, the subsequence of the corpus between (and including) positions i and $i+l$. The pointer is positioned at $i+l+1$, the next segment is sampled and so on. An example of a random segmentation from our experiments is "@he@had@b egu n@to@show @his@cap acity@f" where space was replaced with "@" and the next segment starts with "or@".

The corpus is segmented this way m times (where m is a parameter) and the m random segmentations are concatenated. The unit-context matrix is derived from this concatenated corpus.

Multiple random segmentation has two advantages. First, there is no redundancy since, in any given random segmentation, two ngrams do not overlap and are not subsequences of each other. Second, a single random segmentation would only cover a small part of the space of possible ngrams. For example, a random segmentation of "a rose is a rose is a rose" might be "[a ros][e is a ros][e is][a rose]". This segmentation does not contain the segment "rose" and this part of the corpus can then not be exploited to learn a good embedding for the fourgram "rose". However, with multiple random segmentation, it is likely that this part of the corpus does give rise to the segment "rose" in one of the segmentations and can contribute information to learning a good embedding for "rose".

We took the idea of random segmentation from work on biological sequences (Asgari and Mofrad, 2015; Asgari and Mofrad, 2016). Such sequences have no delimiters, so they are a good model if one believes that delimiter-based segmentation is problematic for text.

3.2 Ngram equivalence classes/Permutation

Form-meaning homomorphism premise. Nonsymbolic representation learning does not preprocess the training corpus by means of tokenization and considers many ngrams that would be ignored in tokenized approaches because they span token boundaries. As a result, the number of ngrams that occur in a corpus is an order of magnitude larger for tokenization-free approaches than for tokenization-based approaches. See supplementary for details.

We will see below that this sparseness impacts performance of nonsymbolic text representation negatively. We address sparseness by defining ngram equivalence classes. All ngrams in an equivalence class receive the same embedding.

The relationship between form and meaning is mostly arbitrary, but there are substructures of the ngram space and the embedding space that are systematically related by homomorphism. In this paper, **we will assume the following homomorphism:**

$$g_1 \sim_\tau g_2 \Leftrightarrow \vec{v}(g_1) \sim_= \vec{v}(g_2)$$

where $g_1 \sim_\tau g_2$ iff $\tau(g_1) = \tau(g_2)$ for string transduction τ and $\vec{v}(g_1) \sim_= \vec{v}(g_2)$ iff $|\vec{v}(g_1) - \vec{v}(g_2)|_2 < \epsilon$.

As a simple example consider a transduction τ that deletes spaces at the beginning of ngrams, e.g., $\tau(\text{@Mercedes}) = \tau(\text{Mercedes})$. This is an example of a meaning-preserving τ since for, say, English, τ will not change meaning. We will propose a procedure for learning τ below.

We define $\sim_=$ as "closeness" – not as identity – because of estimation noise when embeddings are learned. We assume that there are no true synonyms and therefore the direction $g_1 \sim_\tau g_2 \Leftarrow \vec{v}(g_1) \sim_= \vec{v}(g_2)$ also holds. For example, "car" and "automobile" are considered synonyms, but we assume that their embeddings are different because only "car" has the literary sense "chariot". If they were identical, then the homomorphism would not hold since "car" and "automobile" cannot be converted into each other by any plausible meaning-preserving τ.

Learning procedure. To learn τ, we define three templates that transform one ngram into another: (i) replace character a_1 with character a_2,

(ii) delete character a_1 if its immediate predecessor is character a_2, (iii) delete character a_1 if its immediate successor is character a_2. The learning procedure takes a set of ngrams and their embeddings as input. It then exhaustively searches for all pairs of ngrams, for all pairs of characters a_1/a_2, for each of the three templates. When two matching embeddings exist, we compute their cosine. For example, for the operation "delete space before M", an ngram pair from our embeddings that matches is "@Mercedes" / "Mercedes" and we compute its cosine. As the characteristic statistic of an operation we take the average of all cosines; e.g., for "delete space before M" the average cosine is .7435. We then rank operations according to average cosine and take the first N_o as the definition of τ where N_o is a parameter. For characters that are replaced by each other (e.g., 1, 2, 3 in Table 1), we compute the equivalence class and then replace the learned operations with ones that replace a character by the canonical member of its equivalence class (e.g., $2 \rightarrow 1$, $3 \rightarrow 1$).

Permutation premise. Tokenization algorithms can be thought of as assigning a particular function or semantics to each character and making tokenization decisions accordingly; e.g., they may disallow that a semicolon, the character ";", occurs inside a token. If we want to learn representations from the data without imposing such hard constraints, then characters should not have any particular function or semantics. A consequence of this desideratum is that if any two characters are exchanged for each other, this should not affect the representations that are learned. For example, if we interchange space and "A" throughout a corpus, then this should have no effect on learning: what was the representation of "NATO" before, should now be the representation of "N TO". We can also think of this type of permutation as a sanity check: it ensures we do not inadvertantly make use of text preprocessing heuristics that are pervasive in NLP.[2]

Let A be the alphabet of a language, i.e., its set of characters, π a permutation on A, C a corpus and $\pi(C)$ the corpus permuted by π. For example, if $\pi(a) = e$, then all "a" in C are replaced with "e" in $\pi(C)$. **The learning procedure should learn identical equivalence classes on C and $\pi(C)$.** So, if $g_1 \sim_\tau g_2$ after running the learning procedure on C, then $\pi(g_1) \sim_\tau \pi(g_2)$ after running the learning procedure on $\pi(C)$.

This premise is motivated by our desire to come up with a general method that does not rely on specific properties of a language or genre; e.g., the premise rules out exploiting the fact through feature engineering that in many languages and genres, "c" and "C" are related. Such a relationship has to be learned from the data.

3.3 Experiments

We run experiments on C, a 3 gigabyte English Wikipedia corpus, and train word2vec skipgram (W2V, (Mikolov et al., 2013)) and fastText skipgram (FTX, (Bojanowski et al., 2016)) models on C and its derivatives. We randomly generate a permutation π on the alphabet and learn a transduction τ (details below). In Table 2 (left), the columns "method", π and τ indicate the method used (W2V or FTX) and whether experiments in a row were run on C, $\pi(C)$ or $\tau(\pi(C))$. The values of "whitespace" are: (i) ORIGINAL (whitespace as in the original), (ii) SUBSTITUTE (what π outputs as whitespace is used as whitespace, i.e., $\pi^{-1}(\text{" "})$ becomes the new whitespace) and (iii) RANDOM (random segmentation with parameters $m = 50$, $k_{\text{min}} = 3$, $k_{\text{max}} = 9$). Before random segmentation, whitespace is replaced with "@" – this character occurs rarely in C, so that the effect of conflating two characters (original "@" and whitespace) can be neglected. The random segmenter then indicates boundaries by whitespace – unambiguously since it is applied to text that contains no whitespace.

We learn τ on the embeddings learned by W2V on the random segmentation version of $\pi(C)$ (C-RANDOM in the table) as described in §3.2 for $N_o = 200$. Since the number of equivalence classes is much smaller than the number of ngrams, τ reduces the number of distinct character ngrams from 758M in the random segmentation version of $\pi(C)$ (C/D-RANDOM) to 96M in the random segmentation version of $\tau(\pi(C))$ (E/F-RANDOM).

Table 1 shows a selection of the N_o operations. Throughout the paper, if we give examples from $\pi(C)$ or $\tau(\pi(C))$ as we do here, we convert characters back to the original for better readability. The two uppercase/lowercase conversions shown

[2] An example of such an inadvertant use of text preprocessing heuristics is that fastText seems to default to lowercase ngrams if embeddings of uppercase ngrams are not available: when fastText is trained on lowercased text and then applied to uppercased text, it still produces embeddings.

substitution	predeletion	postdeletion
2 →1	/r →r	‡@ →‡
3 →1	@‡ →‡	e@ →e
: →.	@‡ →‡	l@ →l
; →.	@H→H	m@→m
E→e	@I →I	ml →m
C→c		

Table 1: String operations that on average do not change meaning. "@" stands for space. ‡ is the left or right boundary of the ngram.

in the table (E→e, C→c) were the only ones that were learned (we had hoped for more). The postdeletion rule ml→m usefully rewrites "html" as "htm", but is likely to do more harm than good. We inspected all 200 rules and, with a few exceptions like ml→m, they looked good to us.

Evaluation. We evaluate the three models on an entity typing task, similar to (Yaghoobzadeh and Schütze, 2015), but based on an entity dataset released by Xie et al. (2016) in which each entity has been assigned one or more types from a set of 50 types. For example, the entity "Harrison Ford" has the types "actor", "celebrity" and "award winner" among others. We extract mentions from FACC (`http://lemurproject.org/clueweb12/FACC1`) if an entity has a mention there or we use the Freebase name as the mention otherwise. This gives us a data set of 54,334, 6085 and 6747 mentions in train, dev and test, respectively. Each mention is annotated with the types that its entity has been assigned by Xie et al. (2016). The evaluation has a strong cross-domain aspect because of differences between FACC and Wikipedia, the training corpus for our representations. For example, of the 525 mentions in dev that have a length of at least 5 and do not contain lowercase characters, more than half have 0 or 1 occurrences in the Wikipedia corpus, including many like "JOHNNY CARSON" that are frequent in other case variants.

Since our goal in this experiment is to evaluate tokenization-free learning, not tokenization-free utilization, we use a simple utilization baseline, the bag-of-ngram model (see §1). A mention is represented as the sum of all character ngrams that embeddings were learned for. Linear SVMs (Chang and Lin, 2011) are then trained, one for each of the 50 types, on train and applied to dev and test. Our evaluation measure is micro F_1 on all typing decisions; e.g., one typing decision is: "Harrison Ford" is a mention of type "actor". We tune thresholds on dev to optimize F_1 and then use these thresholds on test.

3.4 Results

Results are presented in Table 2 (left). Overall performance of FTX is higher than W2V in all cases. For ORIGINAL, FTX's recall is a lot higher than W2V's whereas precision decreases slightly. This indicates that FTX is stronger in both learning and application: in learning it can generalize better from sparse training data and in application it can produce representations for OOVs and better representations for rare words. For English, prefixes, suffixes and stems are of particular importance, but there often is not a neat correspondence between these traditional linguistic concepts and internal FTX representations; e.g., Bojanowski et al. (2016) show that "asphal", "sphalt" and "phalt" are informative character ngrams of "asphaltic".

Running W2V on random segmentations can be viewed as an alternative to the learning mechanism of FTX, which is based on character ngram cooccurrence; so it is not surprising that for RANDOM, FTX has only a small advantage over W2V.

For C/D-SUBSTITUTE, we see a dramatic loss in performance if tokenization heuristics are not used. This is not surprising, but shows how powerful tokenization can be.

C/D-ORIGINAL is like C/D-SUBSTITUTE except that we artificially restored the space – so the permutation π is applied to all characters except for space. By comparing C/D-ORIGINAL and C/D-SUBSTITUTE, we see that the space is the most important text preprocessing feature employed by W2V and FTX. If space is restored, there is only a small loss of performance compared to A/B-ORIGINAL. So text preprocessing heuristics other than whitespace tokenization in a narrow definition of the term (e.g., downcasing) do not seem to play a big role, at least not for our entity typing task.

For tokenization-free embedding learning on random segmentation, there is almost no difference between original data (A/B-RANDOM) and permuted data (C/D-RANDOM). This confirms that our proposed learning method is insensitive to permutations and makes no use of text preprocessing heuristics.

We achieve an additional improvement by applying the transduction τ. In fact, FTX perfor-

			whitespace	ORIGINAL			SUBSTITUTE			RANDOM		
			measure	P	R	F_1	P	R	F_1	P	R	F_1
method	π	τ										
A W2V	−	−		.538	.566	.552				.525	.596	.558
B FTX	−	−		.530	.628	.575				.528	.608	.565
C W2V	+	−		.535	.560	.547	.191	.296	.233	.514	.605	.556
D FTX	+	−		.530	.623	.573	.335	.510	.405	.531	.608	.567
E W2V	+	+								.503	.603	.548
F FTX	+	+								.551	.618	.582

	query	neighbor	r
1	Abdulaziz	Abdul Azi	2
2	codenamed	code name	1
3	Quarterfi	uarter-Fi	1
4	worldreco	orld-reco	1
5	antibodie	stem cell	1
6	eflectors	ear wheel	1
7	ommandeer	rash land	1
8	reenplays	ripts for	1
9	roughfare	ugh downt	1
10	ilitating	e-to-face	1

Table 2: Left: Evaluation results for named entity typing. Right: Neighbors of character ngrams. Rank $r = 1/r = 2$: nearest / second-nearest neighbor.

mance for F-RANDOM (F_1 of .582) is better than tokenization-based W2V and FTX performance. Thus, our proposed method seems to be an effective tokenization-free alternative to tokenization-based embedding learning.

3.5 Analysis of ngram embeddings

Table 2 (right) shows nearest neighbors of ten character ngrams, for the A-RANDOM space. Queries were chosen to contain only alphanumeric characters. To highlight the difference to symbol-based representation models, we restricted the search to 9-grams that contained a delimiter at positions 3, 4, 5, 6 or 7.

Lines 1–4 show that "delimiter variation", i.e., cases where a word has two forms, one with a delimiter, one without a delimiter, is handled well: "Abdulaziz" / "Abdul Azi", "codenamed" / "code name", "Quarterfinal" / "Quarter-Final", "world-record" / "world-record".

Lines 5–9 are cases of ambiguous or polysemous words that are disambiguated through "character context". "stem", "cell", "rear", "wheel", "crash", "land", "scripts", "through", "downtown" all have several meanings. In contrast, the meanings of "stem cell", "rear wheel", "crash land", "(write) scripts for" and "through downtown" are less ambiguous. A multiword recognizer may find the phrases "stem cell" and "crash land" automatically. But the examples of "scripts for" and "through downtown" show that what is accomplished here is not multiword detection, but a more general use of character context for disambiguation.

Line 10 shows that a 9-gram of "face-to-face" is the closest neighbor to a 9-gram of "facilitating". This demonstrates that form and meaning sometimes interact in surprising ways. Facilitating a meeting is most commonly done face-to-face. It is not inconceivable that form – the shared trigram "fac" or the shared fourgram "faci" in "facilitate" / "facing" – is influencing meaning here in a way that also occurs historically in cases like "ear" 'organ of hearing' / "ear" 'head of cereal plant', originally unrelated words that many English speakers today intuit as one word.

4 Utilization: Tokenization-free representation of text

4.1 Methodology

The main text representation model that is based on ngram embeddings similar to ours is the **bag-of-ngram model**. A sequence of characters is represented by a single vector that is computed as the sum of the embeddings of all ngrams that occur in the sequence. In fact, this is what we did in the entity typing experiment. In most work on bag-of-ngram models, the sequences considered are words or phrases (see (Schuetze, 2016) for citations). In a few cases, the model is applied to longer sequences, including sentences and documents; e.g., (Schütze, 1992), (Wieting et al., 2016).

The basic assumption of the bag-of-ngram model is that sequence information is encoded in the character ngrams and therefore a "bag-of" approach (which usually throws away all sequence information) is sufficient. The assumption is not implausible: for most bags of character sequences, there is only a single way of stitching them together to one coherent sequence, so in that case information is not necessarily lost (although this is likely when embeddings are added). But the assumption has not been tested experimentally.

Here, we propose **position embeddings**, character-ngram-based embeddings that more fully preserve sequence information.[3] The simple idea is to represent each position as the sum of all ngrams that contain that position. When we set

[3]Position embeddings were independently proposed by Kalchbrenner et al. (2016), see Section 3.6 of their paper.

POS		$r=1$	$r=2$	$r=3$	$r=4$	$r=5$
2	e	wealthies	accolades	bestselle	bestselli	Billboard
3	s	estseller	wealthies	bestselli	accolades	bestselle
15	o	fortnight	afternoon	overnight	allowance	Saturdays
16	n	fortnight	afternoon	Saturdays	Wednesday	magazines
23	o	superhero	ntagraphi	adventure	Astonishi	bestselli
24	m	superhero	ntagraphi	anthology	Daredevil	Astonishi
29	o	anthology	paperback	superhero	Lovecraft	tagraphic
30	o	anthology	paperback	tagraphic	Lovecraft	agraphics
34	u	antagraph	agraphics	paperback	hardcover	ersweekly
35	b	ublishing	ublishers	ublicatio	antagraph	aperbacks

Table 3: Nearest ngram embeddings (rank $r \in [1,5]$) of the position embeddings for "POS", the positions 2/3 (best), 15/16 (monthly), 23/24 (comic), 29/30 (book) and 34/35 (publications) in the Wikipedia excerpt "best-selling monthly comic book publications sold in North America"

$k_{\text{min}} = 3$, $k_{\text{max}} = 9$, this means that the position is the sum of $(\sum_{3 \leq k \leq 9} k)$ ngram embeddings (if all of these ngrams have embeddings, which generally will be true for some, but not for most positions). A sequence of n characters is then represented as a sequence of n such position embeddings.

4.2 Experiments

We again use the embeddings corresponding to A-RANDOM in Table 2. We randomly selected 2,000,000 contexts of size 40 characters from Wikipedia. We then created a noise context for each of the 2,000,000 contexts by replacing one character at position i ($15 \leq i \leq 25$, uniformly sampled) with space (probability $p = .5$) or a random character otherwise. Finally, we selected 1000 noise contexts randomly and computed their nearest neighbors among the 4,000,000 contexts (excluding the noise query). We did this in two different conditions: for a bag-of-ngram representation of the context (sum of all character ngrams) and for the concatenation of 11 position embeddings, those between 15 and 25. Our evaluation measure is mean reciprocal rank of the clean context corresponding to the noise context. This simulates a text denoising experiment: if the clean context has rank 1, then the noisy context can be corrected.

Table 4 shows that sequence-preserving position embeddings perform better than bag-of-ngram representations.

	bag-of-ngram	position embeddings
MRR	.64	.76

Table 4: Mean reciprocal rank of text denoising experiment for bag-of-ngram text representation and position embedding text representation

	exchange@f (in exchange for)	ic@exchang (many contexts)	ing@exchan (many contexts)
exchange@f	1.000	0.008	-0.056
ic@exchang	0.008	1.000	0.108
ing@exchan	-0.056	0.108	1.000
	xchange@ra (exchange rates)	ival@rates (survival rates)	rime@rates (crime rates)
xchange@ra	1.000	0.036	0.050
ival@rates	0.036	1.000	0.331
rime@rates	0.050	0.331	1.000

Table 6: Cosine similarity of ngrams that cross word boundaries and disambiguate polysemous words. The tables show three disambiguating ngrams for "exchange" and "rates" that have different meanings as indicated by low cosine similarity. In phrases like "floating exchange rates" and "historic exchange rates", disambiguating ngrams overlap. Parts of the word "exchange" are disambiguated by preceding context (ic@exchang, ing@exchan) and parts of "exchange" provide context for disambiguating "rates" (xchange@ra).

Table 5 shows an example of a context in which position embeddings did better than bag-of-ngrams, demonstrating that sequence information is lost by bag-of-ngram representations, in this case the exact position of "Seahawks".

Table 3 gives further intuition about the type of information position embeddings contain, showing the ngram embeddings closest to selected position embeddings; e.g., "estseller" (the first 9-gram on the line numbered 3 in the table) is closest to the embedding of position 3 (corresponding to the first "s" of "best-selling"). The kNN search space is restricted to alphanumeric ngrams.

5 Discussion

Single vs. multiple segmentation. The motivation for multiple segmentation is *exhaustive cov-*

		rep. space	similarity	r	left context	center	right context
1	correct				‖s and Seattle S	‖eahawks th	‖at led to publi‖
2	noise (query)				‖s and Seattle S	‖eahawks t	‖at led to publi‖
3		position-emb	.761	1	‖s and Seattle S	‖eahawks th	‖at led to publi‖
4		bag-of-ngram	.904	1	‖arted 15 games	‖fsr the Se	‖ahawks, leading‖
5		bag-of-ngram	.864	6	‖s and Seattle S	‖eahawks th	‖at led to publi‖

Table 5: Illustration of the result in Table 4. "rep. space" = "representation space". We want to correct the error in the corrupted "noise" context (line 2) and produce "correct" (line 1). The nearest neighbor to line 2 in position-embedding space is the correct context (line 3, $r = 1$). The nearest neighbor to line 2 in bag-of-ngram space is incorrect (line 4, $r = 1$) because the precise position of "Seahawks" in the query is not encoded. The correct context in bag-of-ngram space is instead at rank $r = 6$ (line 5). "similarity" is average cosine (over eleven position embeddings) for position embeddings.

erage of the space of possible segmentations. An alternative approach would be to attempt to find a *single optimal segmentation*.

Our intuition is that in many cases *overlapping segments contain complementary information*. Table 6 gives an example. Historic exchange rates are different from floating exchange rates and this is captured by the low similarity of the ngrams `ic@exchang` and `ing@exchan`. Also, the meaning of "historic" and "floating" is non-compositional: these two words take on a specialized meaning in the context of exchange rates. The same is true for "rates": its meaning is not its general meaning in the compound "exchange rates". Thus, we need a representation that contains overlapping segments, so that "historic" / "floating" and "exchange" can disambiguate each other in the first part of the compound and "exchange" and "rates" can disambiguate each other in the second part of the compound. A single segmentation cannot capture these overlapping ngrams.

What text-type are tokenization-free approaches most promising for? The reviewers thought that language and text-type were badly chosen for this paper. Indeed, a morphologically complex language like Turkish and a noisy text-type like Twitter would seem to be better choices for a paper on robust text representation.

However, robust word representation methods like FTX are effective for *within-token* generalization, in particular, effective for both complex morphology and OOVs. If linguistic variability and noise only occur on the token level, then a tokenization-free approach has fewer advantages.

On the other hand, the foregoing discussion of *cross-token* regularities and disambiguation applies to well-edited English text as much as it does to other languages and other text-types as the example of "exchange" shows (which is disambiguated by prior context and provides disambiguating context to following words) and as is also exemplified by lines 5–9 in Table 2 (right).

Still, this paper does not directly evaluate the different contributions that within-token character ngram embeddings vs. cross-token character ngram embeddings make, so this is an open question. One difficulty is that few corpora are available that allow the separate evaluation of white-space tokenization errors; e.g., OCR corpora generally do not distinguish a separate class of white-space tokenization errors.

Position embeddings vs. phrase/sentence embeddings. Position embeddings may seem to stand in opposition to phrase/sentence embeddings. For many tasks, we need a fixed length representation of a longer sequence; e.g., sentiment analysis models compute a fixed-length representation to classify a sentence as positive / negative.

To see that position embeddings are compatible with fixed-length embeddings, observe first that, in principle, there is *no difference between word embeddings and position embeddings* in this respect. Take a sequence that consists of, say, 6 words and 29 characters. The initial representation of the sentence has length 6 for word embeddings and length 29 for position embeddings. In both cases, we need a model that reduces the variable length sequence into a fixed length vector at some intermediate stage and then classifies this vector as positive or negative. For example, both word and position embeddings can be used as the input to an LSTM whose final hidden unit activations are a fixed length vector of this type.

So assessing position embeddings is not a question of variable-length vs. fixed-length representations. Word embeddings give rise to variable-length representations too. The question is solely whether the position-embedding representation is

a more effective representation.

A more specific form of this argument concerns architectures that compute fixed-length representations of subsequences on intermediate levels, e.g., CNNs. The difference between position-embedding-based CNNs and word-embedding-based CNNs is that the former have access to a *vastly increased range of subsequences*, including substrings of words (making it easier to learn that "exchange" and "exchanges" are related) and cross-token character strings (making it easier to learn that "exchange rate" is noncompositional). Here, the questions are: (i) how useful are subsequences made available by position embeddings and (ii) is the increased level of noise and decreased efficiency caused by many useless subsequences worth the information gained by adding useful subsequences.

Independence of training and utilization. We note that our proposed training and utilization methods are completely independent. Position embeddings can be computed from any set of character-ngram-embeddings (including FTX) and our character ngram learning algorithm could be used for applications other than position embeddings, e.g., for computing word embeddings.

Context-free vs. context-sensitive embeddings. Word embeddings are context-free: a given word w like "king" is represented by the same embedding independent of the context in which w occurs. Position embeddings are context-free as well: if the maximum size of a character ngram is k_{max}, then the position embedding of the center of a string s of length $2k_{\text{max}} - 1$ is the same independent of the context in which s occurs.

It is conceivable that text representations could be context-sensitive. For example, the hidden states of a character language model have been used as a kind of nonsymbolic text representation (Chrupala, 2013; Evang et al., 2013; Chrupala, 2014) and these states are context-sensitive. However, such models will in general be a second level of representation; e.g., the hidden states of a character language model generally use character embeddings as the first level of representation. Conversely, position embeddings can also be the basis for a context-sensitive second-level text representation. We have to start somewhere when we represent text. Position embeddings are motivated by the desire to provide a representation that can be computed easily and quickly (i.e., without taking context into account), but that on the other hand is much richer than the symbolic alphabet.

Processing text vs. speech vs. images. Gillick et al. (2016) write: "It is worth noting that noise is often added ... to images ... and speech where the added noise does not fundamentally alter the input, but rather blurs it. [bytes allow us to achieve] something like blurring with text." It is not clear to what extent blurring on the byte level is useful; e.g., if we blur the bytes of the word "university" individually, then it is unlikely that the noise generated is helpful in, say, providing good training examples in parts of the space that would otherwise be unexplored. In contrast, the text representation we have introduced in this paper can be blurred in a way that is analogous to images and speech. Each embedding of a position is a vector that can be smoothly changed in every direction. We have showed that the similarity in this space gives rise to natural variation.

Prospects for completely tokenization-free processing. We have focused on whitespace tokenization and proposed a whitespace-tokenization-free method that computes embeddings of higher quality than tokenization-based methods. However, there are many properties of edited text beyond whitespace tokenization that a complex rule-based tokenizer exploits. In a small explorative experiment, we replaced all non-alphanumeric characters with whitespace and repeated experiment A-ORIGINAL for this setting. This results in an F_1 of .593, better by .01 than the best tokenization-free method. This illustrates that there is still a lot of work to be done before we can obviate the need for tokenization.

6 Conclusion

We introduced the first generic text representation model that is completely nonsymbolic, i.e., it does not require the availability of a segmentation or tokenization method that identifies words or other symbolic units in text. This is true for the training of the model as well as for applying it when computing the representation of a new text. In contrast to prior work that has assumed that the sequence-of-character information captured by character ngrams is sufficient, position embeddings also capture sequence-of-ngram information. We showed that our model performs better than prior work on entity typing and text denoising.

References

Ehsaneddin Asgari and Mohammad R. K. Mofrad. 2015. Protvec: A continuous distributed representation of biological sequences. *CoRR*, abs/1503.05140.

Ehsaneddin Asgari and Mohammad R. K. Mofrad. 2016. Comparing fifty natural languages and twelve genetic languages using word embedding language divergence (WELD) as a quantitative measure of language distance. *CoRR*, abs/1604.08561.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *CoRR*, abs/1607.04606.

Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM TIST*, 2(3):27:1–27:27.

Grzegorz Chrupala. 2013. Text segmentation with character-level text embeddings. *CoRR*, abs/1309.4628.

Grzegorz Chrupala. 2014. Normalizing tweets with edit scripts and recurrent neural embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 2: Short Papers*, pages 680–686. The Association for Computer Linguistics.

Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio. 2016. A character-level decoder without explicit segmentation for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. 1990. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407.

Kilian Evang, Valerio Basile, Grzegorz Chrupala, and Johan Bos. 2013. Elephant: Sequence labeling for word and sentence segmentation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1422–1426. ACL.

Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. 2016. Multilingual language processing from bytes. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 1296–1306. The Association for Computational Linguistics.

Thomas Hofmann. 1999. Probabilistic latent semantic indexing. In *SIGIR '99: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 15-19, 1999, Berkeley, CA, USA*, pages 50–57. ACM.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aäron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. *CoRR*, abs/1610.10099.

Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2177–2185.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3111–3119.

Shakir Mohamed. 2011. *Generalised Bayesian matrix factorisation models*. Ph.D. thesis, University of Cambridge, UK.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543. ACL.

Pushpendre Rastogi, Benjamin Van Durme, and Raman Arora. 2015. Multiview LSA: representation learning via generalized CCA. In Rada Mihalcea, Joyce Yue Chai, and Anoop Sarkar, editors, *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 556–566. The Association for Computational Linguistics.

Hinrich Schuetze. 2016. Nonsymbolic text representation. *CoRR*, abs/1610.00479.

Hinrich Schütze. 1992. Word space. In Stephen Jose Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pages 895–902. Morgan Kaufmann.

Karl Stratos, Michael Collins, and Daniel J. Hsu. 2015. Model-based word embeddings from decompositions of count matrices. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1282–1291. The Association for Computer Linguistics.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2016. Charagram: Embedding words and sentences via character n-grams. In Jian Su, Xavier Carreras, and Kevin Duh, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1504–1515. The Association for Computational Linguistics.

Ruobing Xie, Zhiyuan Liu, Jia Jia, Huanbo Luan, and Maosong Sun. 2016. Representation learning of knowledge graphs with entity descriptions. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 2659–2665. AAAI Press.

Yadollah Yaghoobzadeh and Hinrich Schütze. 2015. Corpus-level fine-grained entity typing using contextual information. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 715–725. The Association for Computational Linguistics.

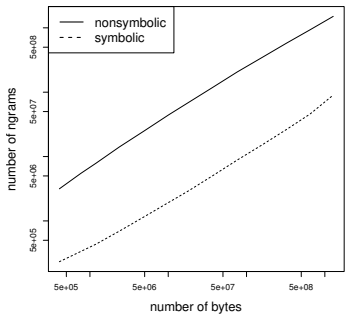

Figure 1: The graph shows how many different character ngrams ($k_{\text{min}} = 3$, $k_{\text{max}} = 10$) occur in the first n bytes of the English Wikipedia for symbolic (tokenization-based) vs. nonsymbolic (tokenization-free) processing. The number of ngrams is an order of magnitude larger in the nonsymbolic approach. We counted all segments, corresponding to $m = \infty$. For the experiments in the paper ($m = 50$), the number of nonsymbolic character ngrams is smaller.

A Supplementary material

A.1 Related work

The related work section appears in the long version of this paper (Schuetze, 2016).

A.2 Acknowledgments

This work was supported by DFG (SCHUE 2246/10-1) and Volkswagenstiftung. We are grateful for their comments to: the anonymous reviewers, Ehsan Asgari, Annemarie Friedrich, Helmut Schmid, Martin Schmitt and Yadollah Yaghoobzadeh.

A.3 Sparseness in tokenization-free approaches

Nonsymbolic representation learning does not preprocess the training corpus by means of tokenization and considers many ngrams that would be ignored in tokenized approaches because they span token boundaries. As a result, the number of ngrams that occur in a corpus is an order of magnitude larger for tokenization-free approaches than for tokenization-based approaches. See Figure 1.

A.4 Experimental settings

W2V hyperparameter settings. size of word vectors: 200, max skip length between words: 5, threshold for occurrence of words: 0, hierarchical softmax: 0, number of negative examples: 5, threads: 50, training iterations: 1, min-count: 5, starting learning rate: .025, classes: 0

FTX hyperparameter settings. learning rate: .05, lrUpdateRate: 100, size of word vectors: 200, size of context window: 5, number of epochs: 1, minimal number of word occurrences: 5, number of negatives sampled: 5, max length of word ngram: 1, loss function: ns, number of buckets: 2,000,000, min length of char ngram: 3, max length of char ngram: 6, number of threads: 50, sampling threshold: .0001

We ran some experiments with more epochs, but this did not improve the results.

A.5 Other hyperparameters

We did not tune $N_o = 200$, but results are highly sensitive to the value of this parameter. If N_o is too small, then beneficial conflations (collapse punctuation marks, replace all digits with one symbol) are not found. If N_o is too large, then precision suffers – in the extreme case all characters are collapsed into one.

We also did not tune $m = 50$, but we do not consider results to be very sensitive to the value of m if it is reasonably large. Of course, if a larger range of character ngram lengths is chosen, i.e., a larger interval $[k_{\text{min}}, k_{\text{max}}]$, then at some point $m = 50$ will not be sufficient and possible segmentations would not be covered well enough in sampling.

The type of segmentation used in multiple segmentation can also be viewed as a hyperparameter. An alternative to random segmentation would be exhaustive segementation, but a naive implementation of that strategy would increase the size of the training corpus by several orders of magnitude. Another alternative is to choose one fixed size, e.g., 4 or 5 (similar to (Schütze, 1992)). Many of the nice disambiguation effects we see in Table 2 (right) and in Table 6 would not be possible with short ngrams. On the other hand, a fixed ngram size that is larger, e.g., 10, would make it difficult to get 100% coverage: there would be positions for which no position embedding can be computed.