On the Importance of Tokenization in Arabic Embedding Models

Mohamed Alkaoud
University of California, Davis
One Shields Ave
Davis, CA 95616
United States
maalkaoud@ucdavis.edu

Mairaj Syed
University of California, Davis
One Shields Ave
Davis, CA 95616
United States
msyed@ucdavis.edu

Abstract

Arabic, like other highly inflected languages, encodes a large amount of information in its morphology and word structure. In this work, we propose two embedding strategies that modify the tokenization phase of traditional word embedding models (Word2Vec) and contextual word embedding models (BERT) to take into account Arabic's relatively complex morphology. In Word2Vec, we segment words into subwords during training time and then compose word-level representations from the subwords during test time. We train our embeddings on Arabic Wikipedia and show that they perform better than a Word2Vec model on multiple Arabic natural language processing datasets while being approximately 60% smaller in size. Moreover, we showcase our embeddings' ability to produce accurate representations of some out-of-vocabulary words that were not encountered before. In BERT, we modify the tokenization layer of Google's pretrained multilingual BERT model by incorporating information on morphology. By doing so, we achieve state of the art performance on two Arabic NLP datasets without pretraining.

1 Introduction

Word embeddings are one of the main building blocks of most natural language processing tasks. Many word embedding techniques have been proposed (Mikolov et al., 2013b; Pennington et al., 2014; Bojanowski et al., 2016) that try to capture better word representations. Although most of the approaches are language agnostic, they were historically designed to be used with English. Nonetheless, it was shown that these techniques work well in other languages (Grave et al., 2018; Soliman et al., 2017).

One feature that is unique to Arabic, and other highly inflected languages, is the expressiveness of its words. The fact that Arabic encodes a large amount of information in its word structure leads to potential problems in learning embeddings due to the large number of forms for each word, the more likely chances of out-of-vocabulary (OOV) instances, and the increase in model size.

In this work, we propose two embedding strategies for Arabic that take into consideration its rich morphology by modifying the tokenization phase. The first technique concerns traditional embedding models and the second a contextual one. Our experiments are done on Word2Vec (Mikolov et al., 2013a) and BERT (Devlin et al., 2018) since they are the most popular traditional and contextual embedding techniques, respectively. Nonetheless, the approaches we propose are embedding-agnostic and can be applied to other embedding techniques. Figure 1 summarizes our two approaches and illustrates what happens in the training and inferences stages of each approach.

In traditional embeddings, we analyze the effect of tokenizing words into subwords by splitting their suffixes and prefixes before training an embedding model and then combining these subwords using an algorithm we propose. We show that by doing so we get:

1. Better performance: our model outperforms Word2Vec in multiple tasks.
2. Smaller size: our model is 59.6% smaller than a Word2Vec model trained on the same corpus.

This work is licensed under a Creative Commons Attribution 4.0 International License. License details: http://creativecommons.org/licenses/by/4.0/.

Proceedings of the Fifth Arabic Natural Language Processing Workshop, pages 119–129
Barcelona, Spain (Online), December 12, 2020

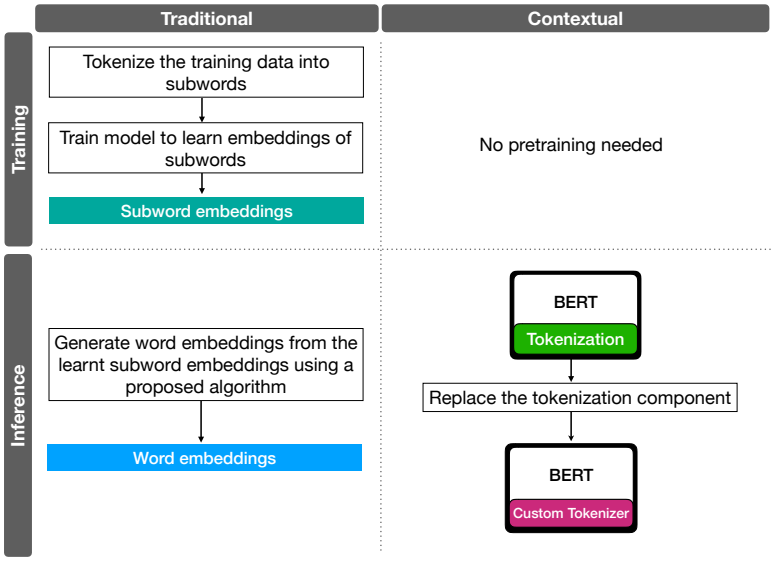

Figure 1: The training and inference stages of our two proposed strategies.

3. Superior out-of-vocabulary handling: our model is able to handle some OOV instances unlike Word2Vec.

In contextual embeddings, we investigate different tokenization schemes when using BERT without needing any pretraining, in contrast to previous approaches such as Antoun et al. (2020). We simply modify the tokenization part of Google's pretrained multilingual BERT model (Devlin et al., 2018) resulting in models that:

1. Achieve state of the art results on two Arabic NLP datasets.
2. Do not require pretraining and can work on top of existing models.

The rest of the paper is structured as follows: Section 2 introduces the background and explains some of the fundamental ideas of word embeddings; Section 3 defines the process of gathering and cleaning our data; Section 4 highlights the embedding approaches we are proposing; Section 5 details the experiments and the results; Section 6 discusses related work; and Section 7 concludes by summarizing our findings and pointing to potential directions for future work.

2 Background

Word embedding techniques rely on the distributional hypothesis (Harris, 1954) which suggests that words that appear in similar contexts tend to have similar meanings. Mikolov et al. (2013a) popularized word embeddings when they introduced Word2Vec and showed that it produces representations that capture not only syntax but also words' semantics. Many related techniques have been produced after that (Mikolov et al., 2013b; Pennington et al., 2014; Bojanowski et al., 2016; Joulin et al., 2016). The word vectors produced by such techniques capture interesting semantics; one popular example is how the the vectors capture relationships between them. For example, if we subtract the vector for 'man' from the vector of 'king', and then add the vector of 'woman' we get very close to the vector of 'queen'.

Contextual word embeddings were proposed (Devlin et al., 2018; Peters et al., 2018; Liu et al., 2019) to tackle the issues that arise from words have multiple senses and meanings. In traditional embeddings such as Word2Vec, each word is encoded in a vector, which is a fixed representation. This may cause problems with homographs and words that have multiple senses depending on the context. For example the wear 'bear' means two very different things in the following sentences: "The right of the people to keep and *bear* arms shall not be infringed." and "A wild *bear* was seen in the city." Yet, traditional embeddings will only capture one fixed representation. Contextual word embeddings aim to solve this issue by modeling embeddings where the context of the word will affect its generated representation.

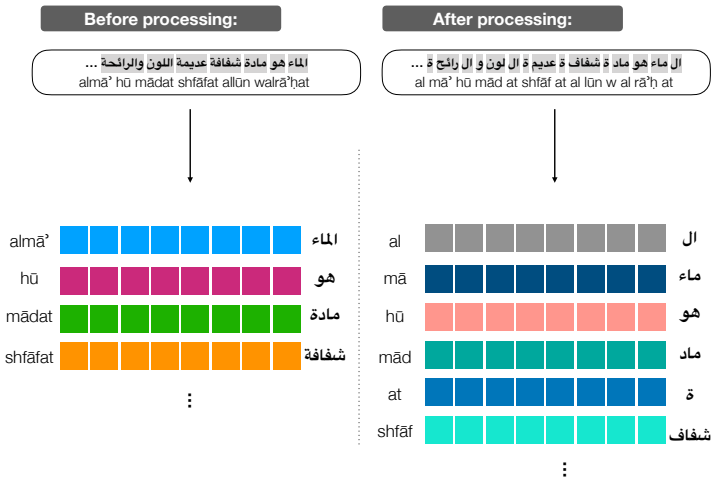

Figure 2: The effect of word segmentation on the resulting vectors. In the top we have representations of words and in the bottom we have representations of subwords.

Verb	Forms
go	go, went, going, gone, goes
ذهب	ذهب، يذهب، سيذهب، ذهبا، يذهبا، سيذهبا، ذهبوا، يذهبوا، سيذهبوا، ذهبنا، نذهب، سنذهب، ذهبت، تذهب، ستذهب، ذهبتا، تذهبا، ...

Table 1: Verb forms in English and Arabic.

3 Data

In this section, we describe the process of gathering and preparing the data used for training the embeddings. We do not require any training data for contextual embeddings since we do not require pretraining: we use Google's multilingual BERT model (Devlin et al., 2018), which supports 104 languages (including Arabic) and was trained on their respective Wikipedia dumps. For traditional embeddings, we use Arabic Wikipedia as a corpus. We downloaded the Wikipedia dump from January 2018 and then cleaned it by using WikiExtractor[1], which is a utility that generates plain text from an XML formatted Wikipedia dump. We then use a custom set of regexes to filter out all non-Arabic words, such as English words and numbers, and remove all diacritics and kashidas resulting in over 86 million tokens.

4 Approach

One example of Arabic's morphological complexity is in the number of verb forms it possesses which is much higher than in English as we can see in Table 1. While traditionally, this aspect of Arabic has been challenging to the natural language processing and computational linguistics communities (Farghaly and Shaalan, 2009; Al-Ayyoub et al., 2018), we asked whether we may benefit from this characteristic. Can we tokenize text differently for Arabic than we do for English, and would that result in better performance on NLP tasks? We experimented with two approaches; one applied to traditional embedding models (Word2Vec) and the other on contextual models (BERT).

4.1 Traditional Embedding Models

Traditional word embedding models are trained on a large corpus of text. The main difference in our approach is that we preprocess the text before feeding it into the embedding algorithm by splitting every word into subwords, which are its prefix(es), stem, and suffix(es) using Farasa, an Arabic segmenter,

[1]https://github.com/attardi/wikiextractor

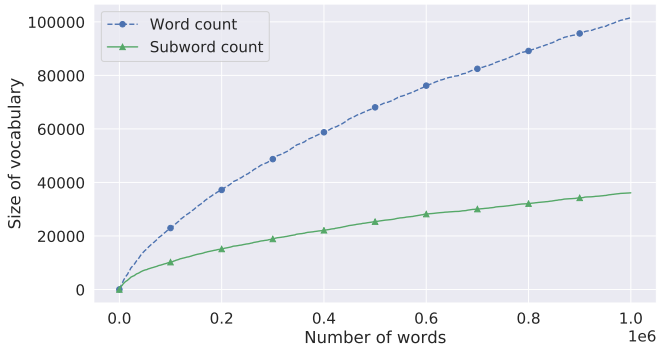

Figure 3: The increase in the size of the vocabulary when using words and subwords. The x-axis shows the number of words processed in the Arabic Wikipedia. The y-axis indicates the size of the vocabulary.

(Abdelali et al., 2016). The effect of using the Farasa segmenter can be seen in Figure 2[2], where each row of squares represents a vector. Then we train the resulting corpus using Word2Vec, though we note that our approach is embedding agnostic and may be used with any embedding model. Notice that our vocabulary, and the resulting vectors, will be completely different now as shown in Figure 2. It may seem at first glance that our vocabulary is increasing, but in fact it decreases as we keep adding more words as shown in Figure 3, which depicts the sizes of the vocabularies in the first million words in Arabic Wikipedia.

One question that comes to mind is how do we generate embeddings for words that were split into subwords because in most cases we want embeddings at the word level and not at the subword level. We propose the following technique for getting the embeddings of all types of words, including those with multiple subwords. For words with only one component, we just return the embeddings learnt by the model. For words with multiple components (subwords), we get the embedding of the longest subword, and the average of the embeddings of the remaining subwords. We then take a weighted average of the two values which results in our embedding as Equations 1 and 2 show.

$$l = \arg\max_{x \in S}(|x|) \tag{1}$$

$$v = \alpha * (M[l]) + (1 - \alpha) * (\sum_{x \in S \setminus \{l\}} \frac{M[x]}{n-1}) \tag{2}$$

where S is the set containing all the subwords contained in the word, M is the learnt model that contains embeddings for all subwords, and α is a parameter that decides the weights between the longest subword and the other subwords. Algorithm 1 illustrates the algorithm used in determining embeddings for all cases.

This method not only allows us to generate embeddings for all words in the original corpus, but also increases its capacity to deal with out-of-vocabulary (OOV) words that the model has never seen before as shown in Figure 4. For example, in Figure 4, we see how our model can produce a representation of 'and their iPhone' which is one word in Arabic. A traditional model trained on the Arabic Wikipedia will fail to produce a representation of the word 'and their iPhone' because that word never appeared in Wikipedia. In fact, since there are many forms for each word, no matter how large the training corpus is, it is almost impossible for it to have seen occurrences of all possible forms of all words in it. Our model can tackle this problem because it operates on a subword level and has seen all the three components that make up the word: 'and', 'their' and 'iPhone' as shown in Figure 4. This procedure allows one to expand a model's vocabulary without retraining or requiring numerous examples of a given word. Of

[2]All transliterations in the paper follow the International Journal of Middle East Studies (IJMES) transliteration system.

Algorithm 1 Generating embeddings of words from subwords

function GETEMBEDDING($word, \alpha, model$)
  if $word \in model$ **then**
    return $model[word]$
  else
    $S =$ get_components$(word)$
    for $s \in S$ **do**
      if $s \notin model$ **then**
        return $error$
    $l = argmax_{s \in S}(|x|)$
    $S = S - l$
    return $\alpha * (model[l]) + (1 - \alpha) * (sum([model[s] \text{ for } s \in S]) \div |S|)$

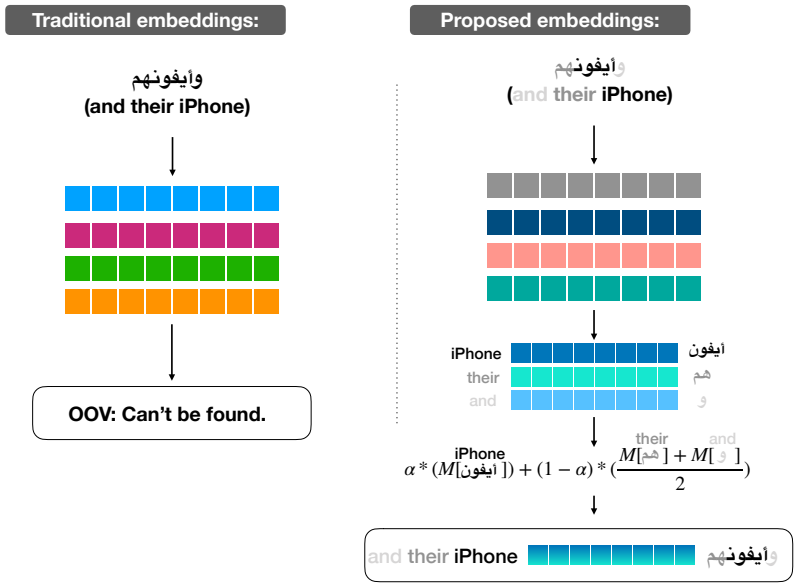

Figure 4: Dealing with out-of-vocabulary words in both the traditional models and our proposed model.

course, not all out-of-vocabulary words will be found this way. Nonetheless, it's a cheap way to generate representations of new words that is not possible in classical approaches.

4.2 Contextual Embedding Models

Transformer-based (Vaswani et al., 2017) contextual embedding models, such as BERT (Devlin et al., 2018), often require a tokenization step that solves problems such as the out-of-vocabulary issue. Byte-pair encoding (BPE) (Sennrich et al., 2015; Gage, 1994), one of the most popular tokenization methods relies on segmenting each word into the most frequent subwords. Shapiro and Duh (2018a) have shown that byte-pair encoding does not perform well for Arabic compared to other languages. One possible explanation of this is that byte-pair encoding does not include information derived from a given language's morphology. We did some experiments using BERT's pretrained tokenizer and found instances where the produced segments generated erroneous meanings. For example, the word *mal'ab* in most cases is a noun that refers to a stadium or field. BERT tokenizes *mal'ab* by segmenting it to *mal* (milliliter) and *'ab* (gulp or fill up), instead of the correct segmentation: *ma* (a prefix used to create nouns of place) and *l'ab* (play). BERT's segmentation seems to indicate that the word *mal'ab* is related to liquids and water (milliliter/gulp/fill up). Keep in mind that both *ma* and *l'ab* are in BERT's subword vocabulary.

We propose two methods that aim to incorporate a language's structure via better segmentations, which

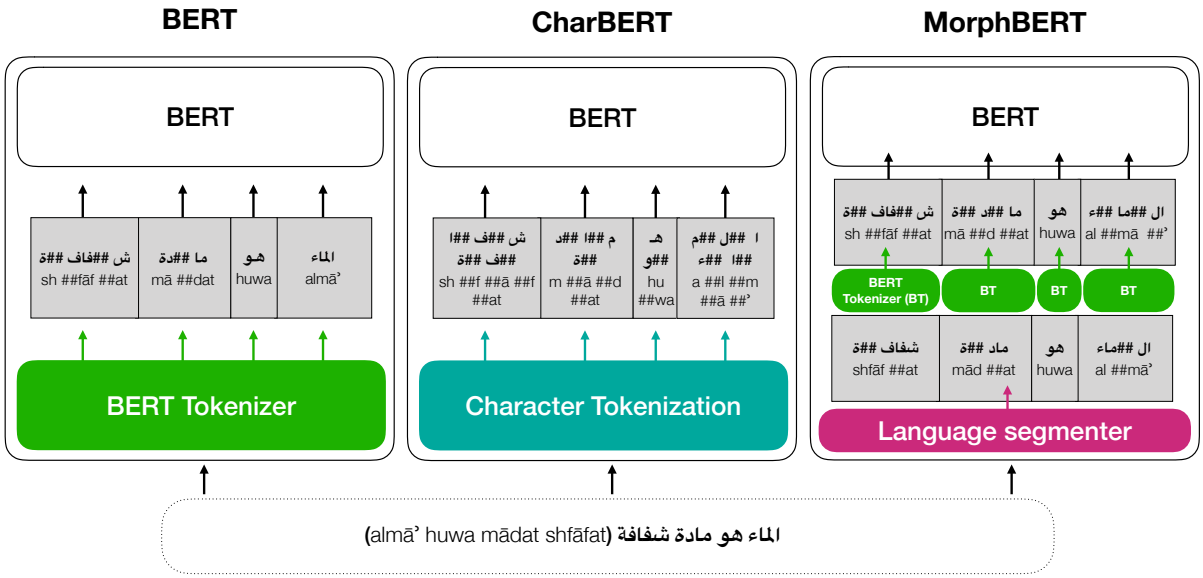

Figure 5: The different segmentation approaches: BERT (default tokenizer), CharBERT, and MorphBERT.

we call MorphBERT (Morphology BERT) and CharBERT (Character BERT). In MorphBERT, a custom tokenizer is used to replace the default tokenizer layer as seen in Figure 5. That custom tokenizer will use a language specific segmenter, Farasa (Abdelali et al., 2016) in our case, to segment each word before processing it. We then pass each word, after segmentation, to the original model's tokenizer to make sure that the produced segments are in the model's vocabulary. Keep in mind that MorphBERT differs from AraBERTv1 (Antoun et al., 2020), an Arabic BERT model that also utilizes Farasa, in that it does not require pretraining. In addition to that, Antoun et al. (2020) preprocess the training corpus by segmenting it using Farasa before training AraBERTv1 which is not the case with MorphBERT.

In CharBERT, we segment everything to characters as shown in Figure 5. The main idea behind CharBERT is to let the network learn these language structures on its own. Both of these models do not require training and can be used with any pretrained model as we see in Figure 5. This is important due to the expensive — money-wise, time-wise, and environment-wise — process of training BERT and other state of the art models. We believe that developing simpler, more sustainable, and more efficient NLP models is of an utmost importance due to the many problems that arise from computationally heavy models, (Strubell et al., 2019) which can take weeks to train on many TPUs/GPUs. Moreover, most people, especially in less developed countries, do not have the resources to train these models, which limits accessibility.

5 Experiments and Results

In this section we detail our experiments and highlight the results produced by the models we proposed.

5.1 Evaluation Datasets

For intrinsic evaluation of traditional embedding models we used the Arabic word analogy benchmark created by Elrazzaz et al. (2017). This dataset consists of nine relations that each consist of over 100 word pairs. We use the following datasets to evaluate our models extrinsically:

1. **APMD**: The Arab poem meters dataset (Alyafeai, 2020) which consists of 55,440 poem verses with each verse classified into one of the fourteen Arabic poetry meters. The data is split into training and testing sets.

Dimension		50	100	200
Skipgram	Base model	5.76%	8.33%	8.88%
	Our model (α=0.3)	5.36%	8.15%	9.40%
CBOW	Base model	6.00%	8.72%	10.05%
	Our model (α=0.3)	**6.31**%	**8.92**%	**10.10**%

Table 2: Top-1 accuracy in the word analogies test.

2. **HARD**: The Hotel Arabic Reviews Dataset (Elnagar et al., 2018) consists of 93,700 hotel reviews that are classified into positive or negative according to their rating. Reviews with a rating of four or five were assigned positive, and those with a rating of one or two were labeled negative. Reviews with a rating of three were ignored. We split the data into 80% and 20% training and testing sets respectively using the script provided by Antoun et al. (2020).
3. **LABR**: The Large-scale Arabic Book Reviews dataset (Aly and Atiya, 2013) consists of 63,000 book reviews rated between one and five. We use the unbalanced two-class dataset, where reviews with a rating of one or two are labeled negative, and those with a rating on four or five are labeled positive. The data is split into training and testing sets.

5.2 Traditional Embedding Models

5.2.1 Intrinsic evaluation

Word analogies, which consists of sets of pairs that share a common relationship, are often used to evaluate different embedding techniques. For example, let's say that we have the following pairs: (king, queen), and (male, female). Both of these pairs contain a word that indicates masculinity and a second word that indicates the feminine version of the first word. A perfect word embedding representation should be able to capture this relationship, and the way to mathematically measure it is by calculating the vector king$-$male$+$female. After that, we check to see whether the resulting vector is the closest to the vector queen or not. We use the Arabic word analogy benchmark created by Elrazzaz et al. (2017) to evaluate our approach. We used Gensim's (Řehůřek and Sojka, 2010) word analogy evaluate function to evaluate the models and set the 'dummy4unknown' flag on so that all tuples of pairs that contain a word (or more) that are not in our vocabulary will get zero accuracy (instead of being skipped), similar to the procedure adopted by Elrazzaz et al. (2017).

We train two models: a vanilla Word2Vec model (base model) and our proposed model on the Arabic Wikipedia dataset mentioned in Section 3. For the base model, we set the window size to be five. Since our model segments words before learning embeddings, we compute the average number of subwords per word in our corpus and adjust the window size by that number. The average number of components per word is around two (1.97); to account for that we set window size in our model to be be ten instead of five. For both, we set the number of epochs to be equal to ten. We experimented with multiple α values for our proposed model and found that setting it to 0.3 achieves the best results. Moreover, to avoid vocabulary size discrepancies, we standardize the vocabulary size before the evaluation by ensuring that our model has the same vocabulary as the base model. To do that, we create a new empty set and go through all vocabulary in the base model and if the word exists in our model (words with only one subword) then we add its representation to the set; otherwise, we decompose it and then add its representation according to equation 2. The results are summarized in Table 2. Our method performs as well, if not better, than the Word2Vec while being around 60% smaller in size. The difference in size come from the vocabulary size which is 155K for our model compared to 383K for the base model. We also notice that CBOW performs better than Skip-gram which is consistent with previous research findings (Elrazzaz et al., 2017).

5.2.2 OOV Handling

One important distinction that separates our model from the base model is its ability to accommodate OOV words. To test the quality of these generated OOV vectors, we go through all the OOV words in our analogy benchmark and generate vectors for the ones we can, i.e. the ones for which we have entries

	Accuracy (without OOV handling)	Accuracy (with OOV handling)
FastText	8.05%	6.44%
Our model	10.10%	13.32%

Table 3: Performance of our model compared to fastText when generating vectors for OOV words.

	APMD	LABR	HARD
Base model	29.95%	86.18%	92.99%
Our model	**37.75%**	**86.21%**	**93.09%**

Table 4: Performance (accuracy) of our model compared to the base model on the three datasets.

for all their respective subwords. The total number of OOV words is 127 and we are able to generate representations for 70 of them covering 55.12% of all the OOV words. After that we follow a similar procedure to the one we have done in the previous subsection and check how the accuracy has been affected. We evaluated the performance of our best performing model (CBOW, dim=200) and found that adding the OOV representations increased the accuracy from 10.10% to 13.32%. To compare our OOV representations, we trained a fastText (Bojanowski et al., 2016) model, a popular embedding approach that can handle OOV, and then calculated the accuracy before adding the OOV entities and after; keep in mind that the vocabulary size in both, fastText and our model, will be the same. FastText achieves 8.05% before adding the OOV entities and 6.44% after adding them as shown in Table 3. Not only did fastText achieve worse gains than our model, it actually performs worse than before which calls to question the accuracy of fastText's OOV embeddings for Arabic.

5.2.3 Extrinsic Evaluation

We evaluate our approach on the three datasets mentioned above. We feed the embeddings of the words to a bidirectional Gated Recurrent Units (GRU) (Cho et al., 2014) network to train it. After that, we evaluate the network on the test set. Table 4 shows the performance of our model compared to the base model. We can see that our model clearly outperforms the base model in one dataset (APMD) and performs slightly better on the two other datasets (LABR and HARD).

5.3 Contextual Embedding Models

We fine-tune the three models: the base cased multilingual BERT, MorphBERT, and CharBERT; and then compare their performances on the three datasets mentioned above. We follow the recommendations from BERT's paper (Devlin et al., 2018) in setting the fine-tuning hyperparameters. We run them all for four epochs in batches of 32 or 16 depending on the lengths of input sequences to avoid memory issues on the GPU. We optimize using the Adam algorithm with a learning rate of $2e^{-5}$, $\beta 1$ = 0.9, and $\beta 2$ = 0.999. We also compare our models to AraBERT (AraBERTv0.1 and AraBERTv1) which consists of two monolingual BERT models trained on Arabic that were proposed by Antoun et al. (2020) and use the results reported by them for LABR and HARD. For APMD, we downloaded their models and fine-tuned them on the task. We also evaluate our models on the cleaned ANERcorp (Benajiba and Rosso, 2007; Antoun et al., 2020) NER dataset. For ANERcorp, we used the script provided by Antoun et al. (2020) in fine-tuning our models. Table 5 shows the performance of the five models on the downstream tasks.

It is interesting that by just changing the tokenization method we can improve BERT's performance in Arabic without retraining. As we can see in Table 5, MorphBERT and CharBERT achieve state of the art performance on LABR and APMD respectively. The previous state of the art model for LABR is the MCE-CNN model proposed by Dahou et al. (2019) which achieves an accuracy of 87.48%. Our models perform better than AraBERT in two tasks even though AraBERT was: 1) trained specifically for Arabic, and 2) trained on a larger Arabic corpus: 24GB of data for AraBERT compared to 4.3GB for the multilingual BERT. Although MultiBERT was trained on over a hundred languages, simply replacing tokenizations allowed us to add language specific information without requiring any training. While normally byte pair encoding learns representations of subwords without paying attention to their

	metric	MultiBERT	MorphBERT	CharBERT	AraBERTv0.1	AraBERTv1
APMD	accuracy	70.65%	71.03%	**84.09%**	70.02%	60.99%
LABR	accuracy	85.85%	**89.87%**	85.89%	85.90%	86.70%
HARD	accuracy	95.96%	95.86%	95.67%	96.20%	**96.20%**
ANERcorp	macro-F1	78.4%	79.86%	71.76%	**89.17%**	88.67%

Table 5: Performance of MorphBERT and CharBERT compared to the multilingual BERT (MultiBERT), AraBERTv0.1, and AraBERTv1

meaning, we can utilize this procedure of breaking words into chunks that make more sense as we saw in the *mal'ab* example before. CharBERT in particular is interesting; one would expect that it will require more time to fine-tune since it only uses characters. Nevertheless, it achieves great performance without requiring more epochs than the other methods. One potential issue with CharBERT is that it results in very long sequences due to the character segmentation approach it follows which lead to more frequent truncations than other models. One potential way to mitigate this is by using new models such as Longformer (Beltagy et al., 2020) that allow longer sequences than BERT.

Previous research (Virtanen et al., 2019; Antoun et al., 2020; Vries et al., 2019; Martin et al., 2020) has shown that a language specific BERT model performs better than a multilingual one. This is the first work, according to our knowledge, that shows that by tweaking a multilingual BERT model one can beat a BERT model trained on a specific language. Natural language processing entered a new era with the advent of pretrained models that do not need to be trained from scratch for every task but can simply be tweaked/fine-tuned instead. Our results shows that it may be possible to only have one multilingual model that can be tweaked instead of learning a pretrained model for every language.

6 Related Work

Many works have noted how sensitivity to Arabic's morphological complexity can result in better performance in standard NLP tasks. However, we do not know of any previous work that has specifically focused on the effect of tokenization on different Arabic embedding models. Antoun et al. (2020) trained an Arabic specific BERT model. They also trained another Arabic BERT model in which they segment the text before training the model and showed that it usually improves performance. Taylor and Brychcín (2018) analyzed morphological relations in Arabic word embeddings. They noted that some morphological features are captured in embeddings representations. Shapiro and Duh (2018b) proposed utilizing subword information in training embeddings to enrich the representations and showed that it improves the performance on word similarity tasks. Salama et al. (2018) investigated morphological-based embeddings and lemma-based embeddings. They utilized part-of-speech information to train their embeddings, similar to Trask et al. (2015), and then build lemma-based embeddings from them by aggregating on different senses of each word first and them combining words that share the same lemma. El-Kishky et al. (2019) tackled the problem of extracting roots of words and proposed an extension to fastText (Bojanowski et al., 2016) that utilize morphemes.

7 Conclusion

We show that tokenization that pays attention to Arabic's morphology can create better traditional and contextual embedding models. Breaking words into subwords in Word2Vec not only leads to an increase in performance and reduction in the vocabulary size and hence the model size, but also provides a simple way to produce good out-of-vocabulary representations. We also show the importance of tokenization in BERT where we were able to achieve impressive performance without requiring any pretraining. One possible future work would be to investigate tokenization's effect in other morphologically rich languages such as Hebrew and Turkish and see if our results can be generalized to other highly inflected languages.

References

Ahmed Abdelali, Kareem Darwish, Nadir Durrani, and Hamdy Mubarak. 2016. Farasa: A fast and furious segmenter for Arabic. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 11–16, San Diego, California, June. Association for Computational Linguistics.

Mahmoud Al-Ayyoub, Aya Nuseir, Kholoud Alsmearat, Yaser Jararweh, and Brij Gupta. 2018. Deep learning for arabic nlp: A survey. *Journal of computational science*, 26:522–531.

Mohamed Aly and Amir Atiya. 2013. LABR: A large scale Arabic book reviews dataset. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 494–498, Sofia, Bulgaria, August. Association for Computational Linguistics.

Zaid Alyafeai. 2020. ARBML. `https://github.com/zaidalyafeai/ARBML`, January.

Wissam Antoun, Fady Baly, and Hazem Hajj. 2020. AraBERT: Transformer-based model for Arabic language understanding. In *Proceedings of the 4th Workshop on Open-Source Arabic Corpora and Processing Tools, with a Shared Task on Offensive Language Detection*, pages 9–15, Marseille, France, May. European Language Resource Association.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv:2004.05150*.

Yassine Benajiba and Paolo Rosso. 2007. Anersys 2.0: Conquering the ner task for the arabic language by combining the maximum entropy with pos-tag information. In *IICAI*, pages 1814–1823.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October. Association for Computational Linguistics.

Abdelghani Dahou, Shengwu Xiong, Junwei Zhou, and Mohamed Abd Elaziz. 2019. Multi-channel embedding convolutional neural network model for arabic sentiment classification. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, 18(4), May.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ahmed El-Kishky, Xingyu Fu, Aseel Addawood, Nahil Sobh, Clare Voss, and Jiawei Han. 2019. Constrained sequence-to-sequence Semitic root extraction for enriching word embeddings. In *Proceedings of the Fourth Arabic Natural Language Processing Workshop*, pages 88–96, Florence, Italy, August. Association for Computational Linguistics.

Ashraf Elnagar, Yasmin S. Khalifa, and Anas Einea, 2018. *Hotel Arabic-Reviews Dataset Construction for Sentiment Analysis Applications*, pages 35–52. Springer International Publishing, Cham.

Mohammed Elrazzaz, Shady Elbassuoni, Khaled Shaban, and Chadi Helwe. 2017. Methodical evaluation of Arabic word embeddings. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 454–458, Vancouver, Canada, July. Association for Computational Linguistics.

Ali Farghaly and Khaled Shaalan. 2009. Arabic natural language processing: Challenges and solutions. *ACM Transactions on Asian Language Information Processing (TALIP)*, 8(4):1–22.

Philip Gage. 1994. A new algorithm for data compression. *C Users Journal*, 12(2):23–38.

Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. 2018. Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Zellig S Harris. 1954. Distributional structure. *Word*, 10(2-3):146–162.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric Villemonte de la Clergerie, Djamé Seddah, and Benoît Sagot. 2020. Camembert: a tasty french language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June. Association for Computational Linguistics.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May. ELRA. `http://is.muni.cz/publication/884893/en`.

Rana Aref Salama, Abdou Youssef, and Aly Fahmy. 2018. Morphological word embedding for arabic. *Procedia computer science*, 142:83–93.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Pamela Shapiro and Kevin Duh. 2018a. Bpe and charcnns for translation of morphology: A cross-lingual comparison and analysis. *arXiv preprint arXiv:1809.01301*.

Pamela Shapiro and Kevin Duh. 2018b. Morphological word embeddings for Arabic neural machine translation in low-resource settings. In *Proceedings of the Second Workshop on Subword/Character LEvel Models*, pages 1–11, New Orleans, June. Association for Computational Linguistics.

Abu Bakr Soliman, Kareem Eissa, and Samhaa R El-Beltagy. 2017. Aravec: A set of arabic word embedding models for use in arabic nlp. *Procedia Computer Science*, 117:256–265.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy, July. Association for Computational Linguistics.

Stephen Taylor and Tomáš Brychcín. 2018. The representation of some phrases in arabic word semantic vector spaces. *Open Computer Science*, 8(1):182–193.

Andrew Trask, Phil Michalak, and John Liu. 2015. sense2vec-a fast and accurate method for word sense disambiguation in neural word embeddings. *arXiv preprint arXiv:1511.06388*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Antti Virtanen, Jenna Kanerva, Rami Ilo, Jouni Luoma, Juhani Luotolahti, Tapio Salakoski, Filip Ginter, and Sampo Pyysalo. 2019. Multilingual is not enough: Bert for finnish.

Wietse de Vries, Andreas van Cranenburgh, Arianna Bisazza, Tommaso Caselli, Gertjan van Noord, and Malvina Nissim. 2019. BERTje: A Dutch BERT Model. *arXiv:1912.09582 [cs]*, December.