Composing Byte-Pair Encodings for Morphological Sequence Classification

Anonymous COLING submission

Abstract

This document contains the instructions for preparing a paper submitted to COLING-2020 or accepted for publication in its proceedings. The document itself conforms to its own specifications, and is therefore an example of what your manuscript should look like. These instructions should be used for both papers submitted for review and for final versions of accepted papers. Authors are asked to conform to all the directions reported in this document.

1 Introduction

Things added to model:

- Adaptive learning rate
- Fine-tuning
- Dropouts
- Trained character representations (?)

Things to add:

- (results) With fine-tuning/without fine-tuning
- (data) Data statistics: number of MSD-tags
- (data) Pick other datasets, try to match dataset sizes better (100k-150k train examples, turkish very small atm)

Since its introduction [CITE:TODO], the transformer model (**?**) has emerged as the dominant architecture for statistical language model, gradually displacing recurrent neural networks, in particular the LSTM and its variants. The transformer owes its success to several factors, including the availability of pretrained models, which effectively yield rich contextual word embeddings. Such embeddings can be used as is (for so-called feature extraction), or the pre-trained models can be fine-tuned to specific applications.

At the same time as transformer models became popular, the tokenization of natural language texts have shifted away from methods explicitly oriented on words or morphemes. Rather, statistical approaches are favoured to splits strings of characters into units which are not necessarily meaningful linguistically, but rather have statistically balanced frequencies. For example, the word "scientifically" may be composed of the tokens: "scient", "ifical", "ly" — here the central token does not correspond to a morpheme. Rather than identifying complete words or morphemes, it aims to find sub-word units occurring significantly often. Typical approaches to composing tokens from sub-token units have focused on combining character n-grams (**?**), while other approaches have looked at splitting words into *roots* and *morphemes*, and then combining them. In this paper, we consider in particular Byte-Pair Encodings (BPE) (**?**), which take another approach. One does not specifically look for either character n-grams or morphs, but rather it aim to split a corpus $\mathcal{C}$ into N tokens, where N is user defined. [EXPAND]

Language	Typology	$\frac{BPE}{word}$	Train	Validation	Test
Arabic-PADT	Fusional	1.39	225494	28089	28801
Czech-CAC	Fusional	1.77	395043	50087	49253
Polish-LFG	Fusional	1.75	104730	13161	13076
Spanish-AnCora	Fusional	1.34	439925	55196	54449
Finnish-TDT	Agglutinative	1.98	161791	19876	20541
Basque-BDT	Agglutinative	1.79	97336	12206	11901
Turkish-IMST	Agglutinative	1.73	46417	5708	5734
Estonian-EDT	Agglutinative	1.86	346986	43434	43825

Table 1: Treebank statistics.

One issue with statistical tokenization is that one is seldom interested in the encoding, but rather in the semantically meaningful units in the original texts. Thus the question of mapping data about the token back to the original text arises.

In this paper we explore how to combine Byte-Pair Encodings from a transformer model to perform sequence classification, which is used in particular in the popular BERT model (**?**). For our purposes, the goal in sequence classification is to assign a label to every word in a sentence. When we are using byte-pair encoding (or similar sub-token representations) we must then find some way of combining the units that build the word, before it is eventually assigned a label. Coming back to our example, we must map the feature-set assigned (depending on the context) to ”scient”, ”ifical”, ”ly”. Then this combined feature set is mapped to a class for the whole word “scientifically”.

To our knownledge, this is a little-studied problem. [citet:TODO] have only brushed the surface by reporting that few differences are found between different methods. In this paper we wish to explore the problem in further detail and identify what effect different methods have on the final performance of a model.

Task: In this paper we further focus on the task of morphological sequence classification. Morphological tagging involves identifying a set of morphological features that a word possesses, such as number, person, case, etc. In many languages, morphological features primarily depend on the affixes of words. However, conversely, the morphological class is not determined by the word affixes, nor the whole word. In many cases, the context of the sentence will affect which class should be assigned to a word.

Data: For the task we will use the Universal Dependencies dataset (**?**) annotated with the UniMorph schema (**?**). We are interested both in the general effects of using different composition methods, and whether some methods favor languages with certain typologies. To explore this, we take a sample of eight languages from the Universal Dependencies dataset. Four of them use a *fusional* morphology, meaning that an affix may be indicative of one or more morphological features. Four of them use an *agglutinative* morphology, meaning that each affix is mapped to one and only one morphological feature.

The fusional languages that we consider are Arabic, Czech, Polish and Spanish, and the agglutinative languages that we consider are Finnish, Basque, Turkish and Estonian. The size of the dataset (**JP:** in original words?) and the average number of BPE tokens per word are shown in Table 1.

The fusional languages where chosen such that two of them (Czech and Polish) have a higher BPE per token ratio than the other two (Arabic and Spanish). We make this choice because one factor that impacts the accuracy obtained by a composition method may be the BPE per token ratio. By having both fusional and agglutinative languages with similar BPE per token ratio we can take this variable into account properly in our analysis.

2 Method

(**JP:** This section should be revised, it’s quite unclear what the model is. It would help to: 1. describe the model in computational order 2. use f consistently to denote the part which can vary in the model.)

In this section we present the model used for sequence classification, the methods that we use to compose BPE embeddings, and how we trained the model.

Transformer model For the task we use the XLMR (**?**) model[1]. XLMR is a masked language model based on the transformer (specifically, RoBERTa (**?**)), and trained on data from 100 different languages, (**JP:** including all the languages in our dataset?) using a shared vocabulary. In this experiment we use XLMR_{base} model. It has 12 encoder layers, 16 (?) attention heads and use 768 dimensions for its hidden size.

2.1 Model

For morphological tagging we use the XLMR base model with a classification module on top. The classification module is an LSTM followed by a two layer linear transformation.(**JP:** Is the LSTM always there? I thought mean and sum were used as well?)

The model that we use to predict morphological features is as follows. For each sentence we extract n BPE embeddings x^0 to x^{n-1}. from $XLMR_{base}$, and then align them to words. We then feed all words which consist of more than one BPE embedding to a function f which combines the BPE embeddings. This produces one embedding per word, which we concatenate with character features generated by a character LSTM. We then pass the BPE features concatenated with the character features to an LSTM to extract contextual features.(**JP:** Again, is this active always?) We pass the LSTM outputs to a linear transformation layer that computes scores for each class in the output. We then use a softmax layer to assign probabilities, and compute the loss accordingly.

An outline of the model is presented in Figure 1, in the outline f represents the different methods we use to combine BPE embeddings.

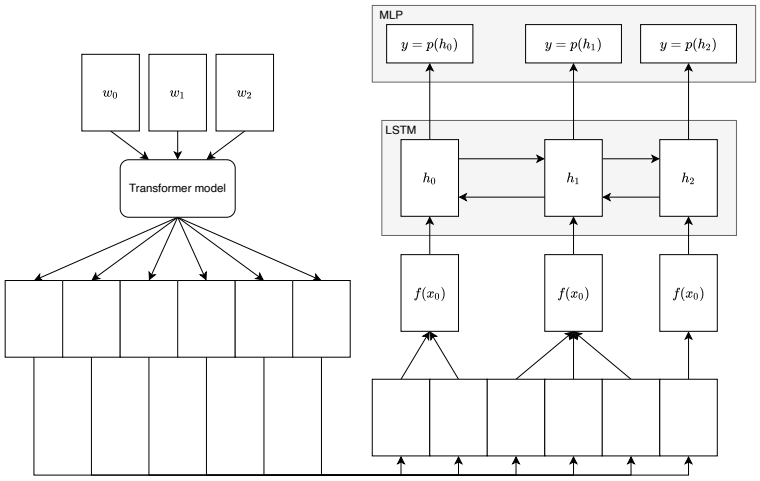

Figure 1: Procedure outline

2.1.1 BPE features

As mentioned previously, we look at methods for composing embeddings of BPE tokens into word embeddings. The XLMR model use 12 layers to compute an embedding for a BPE token, and it has been shown in previous research [CITATIONS, straka, manning, ...] that the different layers of the transformer model encode different types of information. To take advantage of this we compute BPE embeddings as a weighted sum of layer representations. That is, we initialize a parameter from a standard normal distribution w (also tried with a parameter of only 1's, but random was better) of size l, where l is the number of laters in the transformer model. Where r_{ji} is the layer representation at layer j and token position i, we calculate the weighted sum as follows:

[1]We use the huggingface implementation [CITE/LINK]

$$x_i = \sum_{1}^{J} softmax(w)_j r_{ji} \tag{1}$$

Where $softmax(w)_j$ indicate the softmax of the learned importance of layer j. After we have computed weighted sum for each BPE token we proceed to combine them into the tokens as they appear in the data. To align BPE tokens to tokens in the text we use a simple alignment algorithm [It's super simple...].

We look at three method in particular, summation and mean along a dimension and using an RNN. Summation and mean have been used in previous work, but using an RNN have not been explored before to our knowledge.

Sum: For the sum method, we use an element-wise sum. That is, we take the sum for each dimension of the BPE embeddings separately. Thus, for token i we calculate a composite embedding by summing over dimensions $0, \ldots, N$:

$$f(x)_i = \sum_{j=1}^{n} x_i^j \tag{2}$$

Mean: In the mean method we calculate the sum and divide by the number of BPE embeddings in the word. Thus, for token i we calculate a composite embedding by averaging over dimensions $0, \ldots, N$:

$$f(x)_i = \frac{1}{n} \sum_{j=1}^{n} x_i^j \tag{3}$$

RNN: For this method we employ a bidirectional LSTM to compose the BPE embeddings. For each multi-BPE token, we pass the sequence of BPE embeddings through an LSTM and use the final output as the word representation.

2.1.2 Character features

In addition to layer attention we use a character LSTM to extract a word representation based on characters. The final representation that we pass to the (**JP:** Again, this is probably f instead of the LSTM.) word-LSTM is the concatenation of the word representation based on BPE compositions and characters, $w_i = \text{concat}(f(bpe_0, ..., bpe_K), LSTM(c_0, ..., c_M))$

2.1.3 Label smoothing

Given that many of the languages have a large number of morphological tags, we want to prevent the model from growing overconfident for certain classes. To address this issue we introduce label smoothing (**?**), that is, instead of the incorrect classes having 0 probability and the correct class 100% probability we let each of the incorrect classes have a small probability.

Let α be our smoothing value, then given a one-hot encoded target vector t of size (N, C) where N is the number of examples and C the number of classes we calculate the smoothed probabilities as:

$$t_{smooth} = \frac{(1-\alpha)t + \alpha}{C} \tag{4}$$

In esscense, we remove α from the correct class, then assign a probability of $\frac{\alpha}{C}$ to all incorrect classes in t. That is, after subtracting α from the correct class we distribute α uniformly among all classes.

2.2 Training

When fine-tuning the model we freeze the XLMR parameters for the first epoch. When training the model we use cosine annealing learning rate with restarts every epoch, that is, the learning rate starts high then incrementally decrease to $1e-12$ during N steps, where N is the number of batches in an epoch.

Parameter	Value
Epochs	15
Batch size	4 / 32
Character representation size	128
Optimizer	Adam
Learning rate	0.001
Learning rate$_{xlmr}$	1e-06
Label smoothing	0.03

Table 2: Hyperparameters used for training the model. Slashed indicate the value of a parameter when we finetune or extract features.

2.3 Experiments

For completeness, we look at different composition methods both when we fine-tune the transfomer (i.e. when we update the parameters of the transfromer model) and when we only extract features from the model. (maybe not a whole section, might add more things here tho)

3 Results

We present the accuracy of assigning morphological tags to all tokens in Table 3. Both when we finetune and only extract BPE weights we see that using an RNN to compose BPE tokens yields slightly better performance. In general, our results are lower than the reported State-of-the-Art.

		Finetuning			Feature extraction		
Treebank	Baseline	Sum	Mean	RNN	Sum	Mean	RNN
Finnish-TDT	.751	.965	.963	**.967**	.930	.931	**.942**
Basque-BDT	.676	.910	.909	**.920**	.865	.865	**.888**
Turkish-IMST	.620	.898	.891	**.905**	.856	.849	**.866**
Estonian-EDT	.740				.931	.934	**.939**
Spanish-AnCora	.842	.979	.979	**.980**	.968	.967	**.971**
Arabic-PADT	.770	.952	.953	**.954**	.941	.939	**.948**
Czech-CAC	.771				.944	.944	**.952**
Polish-LFG	.657				.907	.907	**.928**

Table 3: Accuracy for morphological tagging. We evaluate both when we finetune the XLMR model and when we only extract BPE embeddings.

		Finetuning			Feature extraction		
Treebank	Baseline	Sum	Mean	RNN	Sum	Mean	RNN
Basque-BDT		0	0	0	0.789	0.780	0.834
Finnish-TDT		0	0	0	0.856	0.847	0.899
Turkish-IMST		0	0	0	0.741	0.735	0.775
Estonian-EDT			0	0	0.856	0.853	0.901
Spanish-AnCora		0	0	0	0.954	0.952	0.962
Arabic-PADT		0	0	0	0.923	0.902	0.936
Czech-CAC		0	0		0.887	0.881	0.924
Polish-LFG		0	0	0	0.844	0.840	0.878

Table 4: Without character LSTM, much more prominent changes in accuracy as expected.

However, the results in Table 3 also include tokens which are only composed of one BPE token. To better evaluate our composition methods we compute the accuracy for tokens which are composed of

two or more BPE tokens. The results can be seen in Table 5. We see again that RNN seem to work better than summation or averaging BPE embeddings.

Given that the number of BPE tokens per text token varies, we also look at the accuracy of the different methods given different nmber of BPE tokens. We show per-language performance with the differnt methods in Figure 2.

	Feature extraction		
Treebank	Sum	Mean	RNN
Finnish-TDT	.893	.897	**.913**
Basque-BDT	.802	.803	**.840**
Turkish-IMST	.807	.796	**.816**
Estonian-EDT	.904	.908	**.916**
Spanish-AnCora	.952	.951	**.959**
Arabic-PADT	.927	.925	**.935**
Czech-CAC	.915	.916	**.930**
Polish-LFG	.834	.833	**.876**

Table 5: Accuracy for morphological tagging on all tokens that are composed of 2 or more BPE tokens.

4 Discussion

Analogously to character embeddings, the RNN method seem to provide stronger results than using either Sum or Mean for combining BPE tokens. We hypothesize that this is because Sum or Mean don't respect the ordering of elements within a word, that is, we don't relate the values of BPE_0 to $BPE_{1:N}$ with respect to the morphological tags.

5 Conclusions

As a general summary of our work, we highlight that language typology is not irrelevant when working with models that employ byte-pair encoding.

References

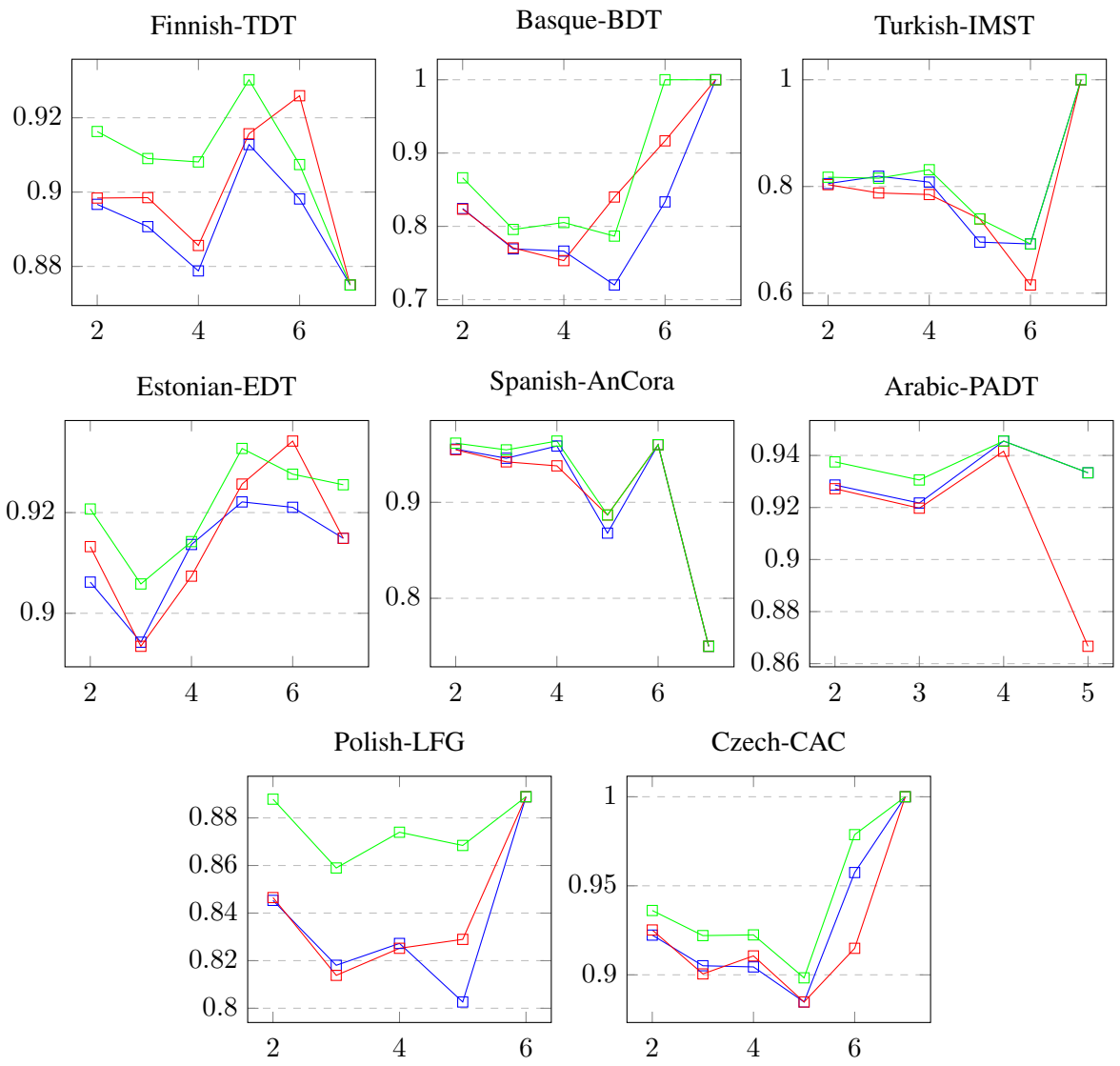

Figure 2: Per language accuracy of tokens composed of more than two BPE tokens. The x-axis indicate how many BPE tokens a word is composed of and the y-axis the accuracy. The different methods are distinguished by color, where Blue is the summation method, red the mean and green RNN.