Reusing Weights in Subword-aware Neural Language Models

Zhenisbek Assylbekov
School of Science and Technology
Nazarbayev University
zhassylbekov@nu.edu.kz

Rustem Takhanov
School of Science and Technology
Nazarbayev University
rustem.takhanov@nu.edu.kz

Abstract

We propose several ways of reusing subword embeddings and other weights in subword-aware neural language models. The proposed techniques do not benefit a competitive character-aware model, but some of them improve the performance of syllable- and morpheme-aware models while showing significant reductions in model sizes. We discover a simple hands-on principle: in a multi-layer input embedding model, layers should be tied consecutively bottom-up if reused at output. Our best morpheme-aware model with properly reused weights beats the competitive word-level model by a large margin across multiple languages and has 20%–87% fewer parameters.

1 Introduction

A statistical language model (LM) is a model which assigns a probability to a sequence of words. It is used in speech recognition, machine translation, part-of-speech tagging, information retrieval and other applications. Data sparsity is a major problem in building traditional n-gram language models, which assume that the probability of a word only depends on the previous n words. To deal with potentially severe problems when confronted with any n-grams that have not explicitly been seen before, some form of smoothing is necessary.

Recent progress in statistical language modeling is connected with neural language models (NLM), which tackle the data sparsity problem by representing words as vectors. Typically this is done twice: at input (to embed the current word of a sequence into a vector space) and at output (to embed candidates for the next word of a sequence). Especially successful are the models in which the architecture of the neural network between input and output is recurrent (Mikolov et al., 2010), which we refer to as recurrent neural network language models (RNNLM).

Tying input and output word embeddings in word-level RNNLM is a regularization technique, which was introduced earlier (Bengio et al., 2001; Mnih and Hinton, 2007) but has been widely used relatively recently, and there is empirical evidence (Press and Wolf, 2017) as well as theoretical justification (Inan et al., 2017) that such a simple trick improves language modeling quality while decreasing the total number of trainable parameters almost two-fold, since most of the parameters are due to embedding matrices. Unfortunately, this regularization technique is not directly applicable to subword-aware neural language models as they receive subwords at input and return words at output. This raises the following questions: Is it possible to reuse embeddings and other parameters in subword-aware neural language models? Would it benefit language modeling quality? We experimented with different subword units, embedding models, and ways of reusing parameters, and our answer to both questions is as follows: There are several ways to reuse weights in subword-aware neural language models, and none of them improve a competitive character-aware model, but some of them do benefit syllable- and morpheme-aware models, while giving significant reductions in model sizes. A simple morpheme-aware model that sums morpheme embeddings of a word benefits most from appropriate weight tying, showing a significant gain over the competitive word-level baseline across different languages and data set sizes. Another contribution of this paper is the discovery of a hands-on principle that in a multi-layer input embedding model, layers should be tied consecutively bottom-up if reused at output.

The source code for the morpheme-aware model is available at `https://github.com/zh3nis/morph-sum`.

Proceedings of NAACL-HLT 2018, pages 1413–1423
New Orleans, Louisiana, June 1 - 6, 2018. ©2018 Association for Computational Linguistics

2 Related Work

Subword-aware NLM: There has been a large number of publications in the last 2–3 years on subword-level and subword-aware NLMs,[1] especially for the cases when subwords are characters (Ling et al., 2015; Kim et al., 2016; Verwimp et al., 2017) or morphemes (Botha and Blunsom, 2014; Qiu et al., 2014; Cotterell and Schütze, 2015). Less work has been done on syllable-level or syllable-aware NLMs (Mikolov et al., 2012; Assylbekov et al., 2017; Yu et al., 2017). For a thorough and up-to-date review of the previous work on subword-aware neural language modeling we refer the reader to the paper by Vania and Lopez (2017), where the authors systematically compare different subword units (characters, character trigrams, BPE, morphs/morphemes) and different representation models (CNN, Bi-LSTM, summation) on languages with various morphological typology.

Tying weights in NLM: Reusing embeddings in word-level neural language models is a technique which was used earlier (Bengio et al., 2001; Mnih and Hinton, 2007) and studied in more details recently (Inan et al., 2017; Press and Wolf, 2017). However, not much work has been done on reusing parameters in subword-aware or subword-level language models. Jozefowicz et al. (2016) reused the *CharCNN* architecture of Kim et al. (2016) to dynamically generate softmax word embeddings without sharing parameters with an input word-embedding sub-network. They managed to significantly reduce the total number of parameters for large models trained on a huge dataset in English (1B tokens) with a large vocabulary (800K tokens) at the expense of deteriorated performance. Labeau and Allauzen (2017) used similar approach to augment the output word representations with subword-based embeddings. They experimented with characters and morphological decompositions, and tried different compositional models (CNN, Bi-LSTM, concatenation) on Czech dataset consisting of 4.7M tokens. They were not tying weights between input and output representations, since their preliminary experiments with tied weights gave worse results.

Our approach differs in the following aspects: we focus on the ways to *reuse* weights at output, seek both model size reduction *and* performance improvement in subword-aware language models, try different subword units (characters, syllables, and morphemes), and make evaluation on small (1M–2M tokens) and medium (17M–51M tokens) data sets across multiple languages.

3 Recurrent Neural Language Model

Let $\mathcal{W}$ be a finite vocabulary of words. We assume that words have already been converted into indices. Let $\mathbf{E}^{\text{in}}_{\mathcal{W}} \in \mathbb{R}^{|\mathcal{W}|\times d_{\mathcal{W}}}$ be an input embedding matrix for words — i.e., it is a matrix in which the wth row (denoted as $\mathbf{w}$) corresponds to an embedding of the word $w \in \mathcal{W}$.

Based on word embeddings $\mathbf{w_{1:k}} = \mathbf{w_1}, \ldots, \mathbf{w_k}$ for a sequence of words $w_{1:k}$, a typical word-level RNN language model produces a sequence of states $\mathbf{h_{1:k}}$ according to

$$\mathbf{h_t} = \text{RNNCell}(\mathbf{w_t}, \mathbf{h_{t-1}}), \quad \mathbf{h_0} = \mathbf{0}. \qquad (1)$$

The last state $\mathbf{h_k}$ is assumed to contain information on the whole sequence $w_{1:k}$ and is further used for predicting the next word w_{k+1} of a sequence according to the probability distribution

$$\Pr(w_{k+1}|w_{1:k}) = \text{softmax}(\mathbf{h_k}\mathbf{E}^{\text{out}}_{\mathcal{W}} + \mathbf{b}), \quad (2)$$

where $\mathbf{E}^{\text{out}}_{\mathcal{W}} \in \mathbb{R}^{d_{\text{LM}}\times|\mathcal{W}|}$ is an output embedding matrix, $\mathbf{b} \in \mathbb{R}^{|\mathcal{W}|}$ is a bias term, and d_{LM} is a state size of the RNN.

Subword-based word embeddings: One of the more recent advancements in neural language modeling has to do with segmenting words at input into subword units (such as characters, syllables, morphemes, etc.) and composing each word's embedding from the embeddings of its subwords. Formally, let $\mathcal{S}$ be a finite vocabulary of subwords,[2] and let $\mathbf{E}^{\text{in}}_{\mathcal{S}} \in \mathbb{R}^{|\mathcal{S}|\times d_{\mathcal{S}}}$ be an input embedding matrix for subwords. Any word $w \in \mathcal{W}$ is a sequence of its subwords $(s_1, s_2, \ldots, s_{n_w}) = \sigma(w)$, and hence can be represented as a sequence of the corresponding subword vectors:

$$[\mathbf{s_1}, \mathbf{s_2}, \ldots, \mathbf{s_{n_w}}]. \qquad (3)$$

A subword-based word embedding model $E(\cdot; \mathbf{E}^{\text{in}}_{\mathcal{S}}, \mathbf{\Theta}^{\text{in}})$ with parameters $(\mathbf{E}^{\text{in}}_{\mathcal{S}}, \mathbf{\Theta}^{\text{in}})$ constructs a word vector $\mathbf{x}$ from the sequence of subword vectors (3), i.e.

$$\mathbf{x} = E(\sigma(w); \mathbf{E}^{\text{in}}_{\mathcal{S}}, \mathbf{\Theta}^{\text{in}}), \qquad (4)$$

[1]*Subword-level* LMs rely on subword-level inputs and make predictions at the level of subwords; *subword-aware* LMs also rely on subword-level inputs but make predictions at the level of words.

[2]As in the case of words, we assume that subwords have already been converted into indices.

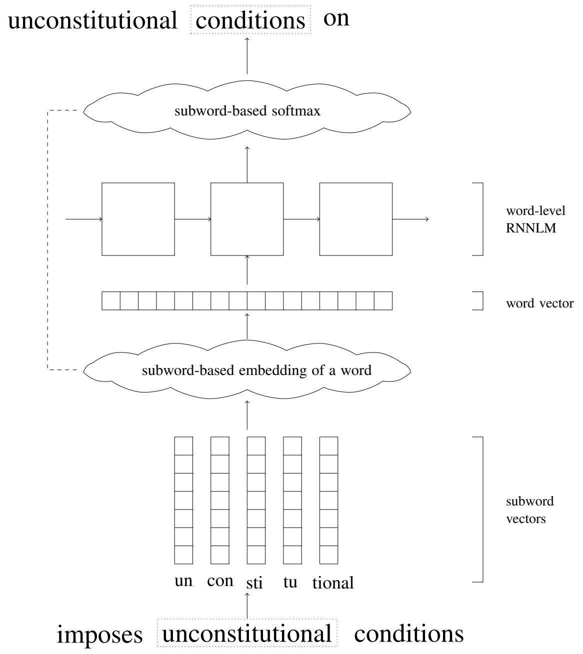
unconstitutional conditions on
subword-based softmax
word-level RNNLM
word vector
subword-based embedding of a word
subword vectors
un con sti tu tional
imposes unconstitutional conditions

Figure 1: Subword-aware RNNLM with subword-based softmax.

which is then fed into a RNNLM (1) instead of a plain embedding $\mathbf{w}$. The additional parameters $\mathbf{\Theta}^{\text{in}}$ correspond to the way the embedding model constructs the word vector: for instance, in the *CharCNN* model of Kim et al. (2016), $\mathbf{\Theta}^{\text{in}}$ are the weights of the convolutional and highway layers.

Reusing word embeddings: Another recent technique in word-level neural language modeling is tying input and output word embeddings:

$$\mathbf{E}^{\text{in}}_{\mathcal{W}} = \left(\mathbf{E}^{\text{out}}_{\mathcal{W}}\right)^T,$$

under the assumption that $d_{\mathcal{W}} = d_{\text{LM}}$. Although being useful for word-level language modeling (Press and Wolf, 2017; Inan et al., 2017), this regularization technique is not directly applicable to subword-aware language models, as they receive subword embeddings at input and return word embeddings at output. In the next section we describe a simple technique to allow reusing subword embeddings $\mathbf{E}^{\text{in}}_{\mathcal{S}}$ as well as other parameters $\mathbf{\Theta}^{\text{in}}$ in a subword-aware RNNLM.

4 Reusing Weights

Let $\mathbf{E}^{\text{out}}_{\mathcal{S}}$ be an output embedding matrix for subwords and let us modify the softmax layer (2) so that it utilizes $\mathbf{E}^{\text{out}}_{\mathcal{S}}$ instead of the word embedding matrix $\mathbf{E}^{\text{out}}_{\mathcal{W}}$. The idea is fairly straightforward: we reuse an embedding model (4) to construct a new word embedding matrix:

$$\hat{\mathbf{E}}^{\text{out}}_{\mathcal{W}} = [E(\sigma(w); \mathbf{E}^{\text{out}}_{\mathcal{S}}, \mathbf{\Theta}^{\text{out}}) \text{ for } w \in \mathcal{W}], \quad (5)$$

and use $\hat{\mathbf{E}}^{\text{out}}_{\mathcal{W}}$ instead of $\mathbf{E}^{\text{out}}_{\mathcal{W}}$ in the softmax layer (2). Such modification of the softmax layer will be referred to as *subword-based softmax*. The overall architecture of a subword-aware RNNLM with subword-based softmax is given in Figure 1. Such a model allows several options for reusing embeddings and weights, which are discussed below.

- Reusing neither subword embeddings nor embedding model weights: As was shown by Jozefowicz et al. (2015), this can significantly reduce the total number of parameters for large models trained on huge datasets (1B tokens) with large vocabularies (800K tokens). However, we do not expect significant reductions on smaller data sets (1-2M tokens) with smaller vocabularies (10-30K tokens), which we use in our main experiments.
- Reusing subword embeddings (*RE*) can be done by setting $\mathbf{E}^{\text{out}}_{\mathcal{S}} = \mathbf{E}^{\text{in}}_{\mathcal{S}}$ in (5). This will give a significant reduction in model size for models with $|\mathbf{E}^{\text{in}}_{\mathcal{S}}| \gg |\mathbf{\Theta}^{\text{in}}|$,[3] such as the morpheme-aware model of Botha and Blunsom (2014).
- Reusing weights of the embedding model (*RW*) can be done by setting $\mathbf{\Theta}^{\text{out}} = \mathbf{\Theta}^{\text{in}}$. Unlike the previous option, this should significantly reduce sizes of models with $|\mathbf{E}^{\text{in}}_{\mathcal{S}}| \ll |\mathbf{\Theta}^{\text{in}}|$, such as the character-aware model of Kim et al. (2016).
- Reusing both subword embeddings and weights of the embedding model (*RE+RW*) can be done by setting $\mathbf{E}^{\text{out}}_{\mathcal{S}} = \mathbf{E}^{\text{in}}_{\mathcal{S}}$ and $\mathbf{\Theta}^{\text{out}} = \mathbf{\Theta}^{\text{in}}$ simultaneously in (5). This should significantly reduce the number of trainable parameters in any subword-aware model. Here we use exactly the same word representations both at input and at output, so this option corresponds to the reusing of plain word embeddings in pure word-level language models.

5 Experimental Setup

Data sets: All models are trained and evaluated on the PTB (Marcus et al., 1993) and the WikiText-2 (Merity et al., 2017) data sets. For the PTB we utilize the standard training (0-20), validation (21-22), and test (23-24) splits along with preprocessing per Mikolov et al. (2010). WikiText-2 is an alternative to PTB, which is approximately two times as large in size and three times as large

[3]$|\mathbf{A}|$ denotes number of elements in $\mathbf{A}$.

in vocabulary (Table 1).

Data set	T	$\|\mathcal{W}\|$	$\|\mathcal{S}\|$	$\|\mathcal{M}\|$
PTB	0.9M	10K	5.9K	3.4K
WikiText-2	2.1M	33K	19.5K	8.8K

Table 1: Corpus statistics. T = number of tokens in training set; $|\mathcal{W}|$ = word vocabulary size; $|\mathcal{S}|$ = syllable vocabulary size; $|\mathcal{M}|$ = morph vocabulary size.

Subword-based embedding models: We experiment with existing representational models which have previously proven effective for language modeling.

- *CharCNN* (Kim et al., 2016) is a character-aware convolutional model, which performs on par with the 2014–2015 state-of-the-art word-level LSTM model (Zaremba et al., 2014) despite having 60% fewer parameters.
- *SylConcat* is a simple concatenation of syllable embeddings suggested by Assylbekov et al. (2017), which underperforms *CharCNN* but has fewer parameters and is trained faster.
- *MorphSum* is a summation of morpheme embeddings, which is similar to the approach of Botha and Blunsom (2014) with one important difference: the embedding of the word itself is not included into the sum. We do this since other models do not utilize word embeddings.

In all subword-aware language models we inject a stack of two highway layers (Srivastava et al., 2015) right before the word-level RNNLM as done by Kim et al. (2016), and the non-linear activation in any of these highway layers is a ReLU. The highway layer size is denoted by d_{HW}.

Word-level RNNLM: There is a large variety of RNN cells to choose from in (1). To make our results directly comparable to the previous work of Inan et al. (2017), Press and Wolf (2017) on reusing word embeddings we select a rather conventional architecture – a stack of two LSTM cells (Hochreiter and Schmidhuber, 1997).

Hyperparameters: We experiment with two configurations for the state size d_{LM} of the word-level RNNLM: 200 (small models) and 650 (medium-sized models). In what follows values outside brackets correspond to small models, and values within brackets correspond to medium models.

- *CharCNN:* We use the same hyperparameters as in the work of Kim et al. (2016), where "large model" stands for what we call "medium-sized model".
- *SylConcat:* $d_{\mathcal{S}} = 50$ (200), $d_{\text{HW}} = 200$ (800). These choices are guided by the work of Assylbekov et al. (2017).
- *MorphSum*: $d_{\mathcal{S}} = d_{\text{HW}} = 200$ (650). These choices are guided by Kim et al. (2016).

Optimizaton method is guided by the previous works (Zaremba et al., 2014; Gal and Ghahramani, 2016) on word-level language modeling with LSTMs. See Appendix A for details.

Syllabification and morphological segmentation: True syllabification of a word requires its grapheme-to-phoneme conversion and then its splitting up into syllables based on some rules. True morphological segmentation requires rather expensive morphological analysis and disambiguation tools. Since these are not always available for under-resourced languages, we decided to utilize Liang's widely-used hyphenation algorithm (Liang, 1983) and an unsupervised morphological segmentation tool, Morfessor 2.0 (Virpioja et al., 2013), as approximations to syllabification and morphological segmentation respectively. We use the default configuration of Morfessor 2.0. Syllable and morpheme vocabulary sizes for both PTB and WikiText-2 are reported in Table 1.

6 Results

In order to investigate the extent to which each of our proposed options benefits the language modeling task, we evaluate all four modifications (no reusing, RE, RW, RE+RW) for each subword-aware model against their original versions and word-level baselines. The results of evaluation are given in Table 2. We have both negative and positive findings which are summarized below.

Negative results:

- The 'no reusing' and RW options should never be applied in subword-aware language models as they deteriorate the performance.
- Neither of the reusing options benefits *CharCNN* when compared to the original model with a plain softmax layer.

Positive results:

- The *RE+RW* option puts *CharCNN*'s performance close to that of the original version, while reducing the model size by 30–75%.
- The *RE* and *RE+RW* are the best reusing options for *SylConcat*, which make it on par with the original *CharCNN* model, despite having 35–75% fewer parameters.

Model	PTB				Wikitext-2			
	Small		Medium		Small		Medium	
	Size	PPL	Size	PPL	Size	PPL	Size	PPL
Word	4.7M	88.1	19.8M	79.8	14M	111.9	50.1M	95.7
Word + reusing word emb's	2.7M	86.6	13.3M	74.5	7.3M	104.1	28.4M	89.9
CharCNN (original)	4.1M	**87.3**	19.4M	**77.1**	8.7M	**101.6**	34.5M	88.7
CharCNN	3.3M	97.5	18.5M	89.2	3.3M	110.6	18.5M	—
CharCNN + RE	3.3M	99.1	18.5M	82.9	3.3M	110.2	18.5M	—
CharCNN + RW	2.2M	93.5	13.6M	103.2	2.2M	111.5	13.6M	—
CharCNN + RE + RW	2.2M	91.0	13.6M	79.9	2.2M	101.8	13.6M	—
SylConcat (original)	3.2M	89.0	18.7M	77.9	8.5M	105.7	36.6M	91.4
SylConcat	1.7M	96.9	17.7M	90.5	3.1M	118.1	23.2M	114.8
SylConcat + RE	1.4M	**87.4**	16.6M	**75.7**	2.1M	**101.0**	19.3M	94.2
SylConcat + RW	1.6M	99.9	15.2M	96.2	2.9M	118.9	19.4M	112.1
SylConcat + RE + RW	1.2M	88.4	12.7M	76.2	1.9M	**101.0**	15.5M	**86.7**
MorphSum (original)	3.5M	87.5	17.2M	78.5	9.3M	101.9	35.8M	90.1
MorphSum	2.4M	89.0	14.5M	82.4	4.5M	100.3	21.7M	86.7
MorphSum + RE	1.6M	85.5	12.3M	74.1	2.8M	97.6	15.9M	81.2
MorphSum + RW	2.2M	89.6	12.8M	81.0	4.4M	101.4	20.0M	86.6
MorphSum + RE + RW	1.5M	**85.1**	10.7M	**72.2**	2.6M	**96.5**	14.2M	**77.5**

Table 2: Results. The pure word-level models and original versions of subword-aware models (with regular softmax) serve as baselines. Reusing the input embedding architecture at output in *CharCNN* leads to prohibitively slow models when trained on WikiText-2 ($\approx$800 tokens/sec on NVIDIA Titan X Pascal); we therefore abandoned evaluation of these configurations.

- The *RE* and *RE+RW* configurations benefit *MorphSum* making it not only better than its original version but also better than all other models and significantly smaller than the word-level model with reused embeddings.

In what follows we proceed to analyze the obtained results.

6.1 CharCNN is biased towards surface form

We hypothesize that the reason *CharCNN* does not benefit from tied weights is that CNN over character embeddings is an excessively flexible model which learns to adapt to a surface form more than to semantics. To validate this hypothesis we pick several words[4] from the English PTB vocabulary and consider their nearest neighbors under cosine similarity as produced by the medium-sized models (with the regular softmax layer) at input (Table 3). As we can see from the examples, the *CharCNN* model is somewhat more biased towards surface forms at input than *SylConcat* and *MorphSum*.[5] When *CharCNN* is reused to generate a softmax embedding matrix this bias is propagated to output embeddings as well (Table 3).

6.2 Tying weights bottom-up

From Table 2 one can notice that tying weights without tying subword embeddings (*RW*) *always* results in worse performance than the tying both weights and embeddings (*RE+RW*). Recall that subword embedding lookup is done before the weights of subword-aware embedding model are used (see Figure 1). This leads us to the following

Conjecture. *Let* $\mathbf{E}_S^{in} = \mathbf{\Theta}_0^{in}, \mathbf{\Theta}_1^{in}, \mathbf{\Theta}_2^{in}, \dots, \mathbf{\Theta}_n^{in}$ *be the parameters of the consecutive layers of a subword-aware input embedding model (4), i.e.* $\mathbf{x} = \mathbf{x}^{(n)} = f_n\left(\mathbf{x}^{(n-1)}; \mathbf{\Theta}_n^{in}\right)$, $\dots$, $\mathbf{x}^{(1)} = f_1\left(\mathbf{x}^{(0)}; \mathbf{\Theta}_1^{in}\right)$, $\mathbf{x}^{(0)} = f_0\left(\sigma(w); \mathbf{E}_S^{in}\right)$ *and let* $\mathbf{E}_S^{out} = \mathbf{\Theta}_0^{out}, \mathbf{\Theta}_1^{out}, \mathbf{\Theta}_2^{out}, \dots, \mathbf{\Theta}_n^{out}$ *be the parameters of the consecutive layers of a subword-aware embedding model used to generate the output projection matrix (5). Let A be a subword-aware neu-*

[4]We pick the same words as Kim et al. (2016).

[5]A similar observation for character-aware NLMs was made by Vania and Lopez (2017).

	Model	In Vocabulary					Out-of-Vocabulary		
		while	*his*	*you*	*richard*	*trading*	*computer-aided*	*misinformed*	*looooook*
INPUT EMBEDDINGS	CharCNN (original)	chile	hhs	god	graham	traded	computer-guided	informed	look
		whole	its	we	harold	tradition	computerized	performed	looks
		meanwhile	her	your	edward	heading	computer-driven	formed	looking
		although	this	i	ronald	eroding	black-and-white	confirmed	looked
	SylConcat (original)	although	my	kemp	thomas	printing	computer-guided	reinforced	—
		though	historic	welch	robert	working	computer-driven	surprised	—
		when	your	i	stephen	lending	computerized	succeeding	—
		mean	irish	shere	alan	recording	computer	succeed	—
	MorphSum (original)	although	mystery	i	stephen	program-trading	cross-border	informed	nato
		whenever	my	ghandi	leonard	insider-trading	bank-backed	injured	lesko
		when	whiskey	we	william	relations	pro-choice	confined	imo
		1980s	sour	cadillac	robert	insurance	government-owned	formed	swapo
I/O EMB'S	CharCNN + RE + RW	thi	her	we	gerard	trades	computer-guided	informed	look
		when	its	your	gerald	trader	large-scale	performed	outlook
		after	the	young	william	traders	high-quality	outperformed	looks
		above	heir	why	edward	trade	futures-related	confirmed	looked

Table 3: Nearest neighbors based on cosine similarity. We underline character ngrams in words which are close to the given word orthographically rather than semantically. The pyphen syllabifier, which is used in *SylConcat*, failed to segment the word 'looooook' into syllables, and therefore its neighbors are not available.

ral language model in which the first $(j+1)$ layers of input and output embedding sub-networks have tied weights:

$$\forall i = \overline{0, j}: \quad \Theta_i^{in} = \Theta_i^{out},$$

and let B be a model in which at least one layer below the $(j+1)^{th}$ layer has untied weights:

$$\exists i = \overline{0, j-1}: \quad \Theta_i^{in} \neq \Theta_i^{out},\ \Theta_j^{in} = \Theta_j^{out}.$$

Then model B performs at most as well as model A, i.e. $PPL_A \leq PPL_B$.

To test this conjecture empirically, we conduct the following experiments: in all three embedding models (*CharCNN*, *SylConcat*, and *MorphSum*), we reuse different combinations of layers. If an embedding model has n layers, there are 2^n ways to reuse them, as each layer can either be tied or untied at input and output. However, there are two particular configurations for each of the embedding models that do not interest us: (i) when neither of the layers is reused, or (ii) when only the very first embedding layer is reused. Hence, for each model we need to check $2^n - 2$ configurations. For faster experimentation we evaluate only small-sized models on PTB. The results are reported in Table 4. As we can see, the experiments in general reject our conjecture: in *SylConcat* leaving an untied first highway layer between tied embedding and second highway layers (denote this as HW$_2$+Emb) turned out to be slightly better than tying all three layers (HW$_2$+HW$_1$+Emb). Recall, that a highway is a weighted average between nonlinear and identity transformations of the incoming vector:

$$\mathbf{x} \mapsto \mathbf{t} \odot \mathrm{ReLU}(\mathbf{xA} + \mathbf{b}) + (\mathbf{1} - \mathbf{t}) \odot \mathbf{x},$$

where $\mathbf{t} = \sigma(\mathbf{xW} + \mathbf{c})$ is a transform gate, $\mathbf{A}$, $\mathbf{W}$, $\mathbf{b}$ and $\mathbf{c}$ are trainable parameters, and $\odot$ is the element-wise multiplication operator. To find out why leaving an untied highway below a tied one is beneficial in *SylConcat*, we compare the distributions of the transform gate values $\mathbf{t}$ from the first highway layers of both configurations, HW$_2$+Emb and HW$_2$+HW$_1$+Emb, in *SylConcat* and *MorphSum* (Figure 2).

We can see that *SylConcat* heavily relies on nonlinearity in the first highway layer, while *MorphSum* does not utilize much of it. This means that in MorphSum, the highway is close to an identity operator ($\mathbf{t} \approx \mathbf{0}$), and does not transform the sum of morpheme vectors much, either at input or at output. Therefore, tying the first highway layer is natural to *Morh-Sum*. *SylConcat*, on the other hand, applies non-linear transformations to the concatenation of syllable vectors, and hence makes additional preparations of the word vector for the needs of the RNNLM at input and for Softmax prediction at output. These needs differ from each other (as shown in the next subsection). This is why *SylConcat* benefits from an additional degree of freedom when the first highway is left untied.

Despite not being true in all cases, and due to being true in many cases, we believe that the above-mentioned conjecture is still useful. In short it can be summarized as a practical hands-

HW_2	HW_1	CNN	Emb	PPL
		✓		94.1
		✓	✓	**92.8**
	✓			94.6
	✓		✓	94.5
	✓	✓		93.1
	✓	✓	✓	**90.1**
✓				94.9
✓			✓	99.2
✓		✓		94.1
✓		✓	✓	92.5
✓	✓			94.3
✓	✓		✓	97.8
✓	✓	✓		96.3
✓	✓	✓	✓	**91.0**

HW_2	HW_1	Emb	PPL
	✓		95.4
	✓	✓	**87.4**
✓			99.0
✓		✓	**87.9**
✓	✓		96.2
✓	✓	✓	88.4

HW_2	HW_1	Emb	PPL
	✓		90.0
	✓	✓	**84.7**
✓			89.9
✓		✓	85.7
✓	✓		89.4
✓	✓	✓	**85.1**

Table 4: Reusing different combinations of layers in small *CharCNN* (left), small *SylConcat* (top right) and small *MorphSum* on PTB data. "✓" means that the layer is reused at output.

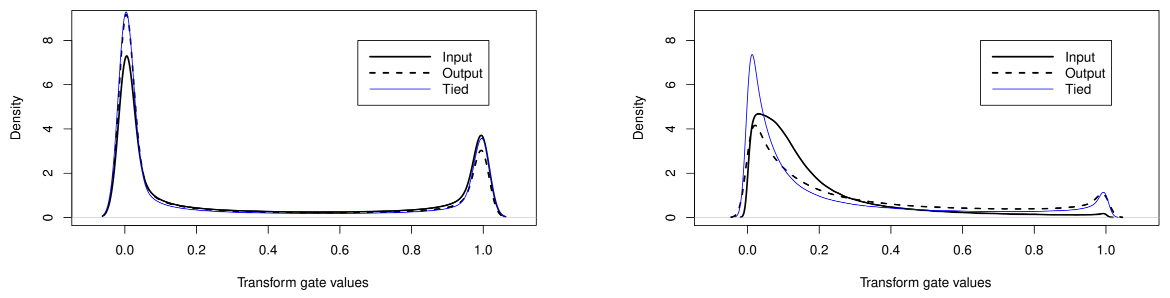

Figure 2: Kernel density estimations of the transform gate values of the first highway layers in *SylConcat* (left) and *MorphSum*. Values corresponding to 'Input' and 'Output' curves come from the HW_2+Emb configurations, while those corresponding to 'Tied' curves come from the HW_2+HW_1+Emb configurations.

on rule:

Layers should be tied consecutively bottom-up,

i.e. one should not leave untied layer(s) below a tied one. Keep in mind that this rule does not guarantee a performance increase as more and more layers are tied. It only says that leaving untied weights below the tied ones is likely to be worse than not doing so.

6.3 Difference between input and output embeddings

One can notice from the results of our experiments (Table 4) that having an untied second highway layer above the first one always leads to better performance than when it is tied. This means that there is a benefit in letting word embeddings slightly differ at input and output, i.e. by specializing them for the needs of RNNLM at input and of Softmax at output. This specialization is quite natural, as input and output representations of words have two different purposes: input representations send a signal to the RNNLM about the current word in a sequence, while output representations are needed to predict the next word given *all* the preceding words. The difference between input and output word representations is discussed in greater detail by Garten et al. (2015) and Press and Wolf (2017). Here we decided to verify the difference indirectly: we test whether intrinsic dimensionality of word embeddings significantly differs at input and output. For this, we apply principal component analysis to word embeddings produced by all models in "no reusing" mode. The results are given in Figure 3, where we can see that dimensionalities of input and output embeddings differ in the word-level model, *CharCNN*, and *SylConcat* models, but the difference is less significant in *MorphSum* model. Interestingly, in word-level and *MorphSum* models the output embeddings have more principal components than the input ones. In *CharCNN* and *SylConcat*, however, results are to other way around. We defer the study of this phenomenon to the future work.

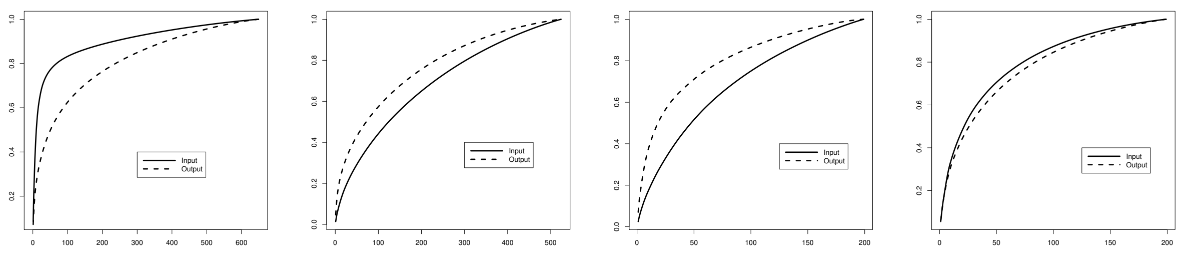

Figure 3: PCA applied to input and word embeddings produced by different models. Horizontal axis corresponds to number of principal components, vertical axis corresponds to percentage of total variance to retain. From left to right: word-level model, *CharCNN*, *SylConcat*, *MorphSum*.

6.4 CharCNN generalizes better than MorphSum

One may expect larger units to work better than smaller units, but smaller units to generalize better than larger units. This certainly depends on how one defines generalizability of a language model. If it is an ability to model unseen text with *unseen* words, then, indeed, character-aware models may perform better than syllable- or morpheme-aware ones. This can be partially seen from Table 3, where the OOV words are better handled by *CharCNN* in terms of in-vocabulary nearest neighbors. However, to fully validate the above-mentioned expectation we conduct additional experiments: we train two models, *CharCNN* and *MorphSum*, on PTB and then we evaluate them on the test set of Wikitext-2 (245K words, 10K word-types). Some words in Wikitext-2 contain characters or morphemes that are not present in PTB, and therefore such words cannot be embedded by *CharCNN* or *MorphSum* correspondingly. Such words were replaced by the <unk> token, and we call them new OOVs[6]. The results of our experiments are reported in Table 5. Indeed, *CharCNN* faces less OOVs on unseen text, and thus generalizes better than *MorphSum*.

Model	# new OOVs	PPL
CharCNN + RE + RW	3659	306.8
MorphSum + RE + RW	4195	316.2

Table 5: Training on PTB and testing on Wikitext-2.

6.5 Performance on non-English Data

According to Table 2, *MorphSum+RE+RW* comfortably outperforms the strong baseline *Word+RE* (Inan et al., 2017). It is interesting to see whether this advantage extends to non-English languages which have richer morphology. For this purpose we conduct evaluation of both models on small (1M tokens) and medium (17M–51M tokens) data in five languages (see corpora statistics in Appendix B). Due to hardware constraints we only train the small models on medium-sized data. We used the same architectures for all languages and did not perform any language-specific tuning of hyperparameters, which are specified in Appendix A. The results are provided in Table 6. As one can see, the advantage of the morpheme-aware model over the word-level one is even more pronounced for non-English data. Also, we can notice that the gain is larger for small data sets. We hypothesize that the advantage of *MorphSum+RE+RW* over *Word+RE* diminishes with the decrease of type-token ratio (TTR). A scatterplot of PPL change versus TTR (Figure 4) supports this hypothesis. Moreover, there is a strong correlation between these two quantities: $\hat{\rho}(\Delta\text{PPL}, \text{TTR}) = 0.84$, i.e. one can predict the mean decrease in PPL from the TTR of a text with a simple linear regression:

$$\Delta\text{PPL} \approx 2,109 \times \text{TTR}.$$

	Model	FR	ES	DE	CS	RU	
S	Word+RE	218	205	305	514	364	D-S
S	MorphSum+RE+RW	**188**	**171**	**246**	**371**	**237**	D-S
M	Word+RE	205	193	277	488	351	D-S
M	MorphSum+RE+RW	**172**	**157**	**222**	**338**	**210**	D-S
S	Word+RE	167	149	285	520	267	D-M
S	MorphSum+RE+RW	**159**	**143**	**242**	**463**	**229**	D-M

Table 6: Evaluation on non-English data. *MorphSum+RE+RW* has significantly less parameters than *Word+RE* (Appendix B). S — small model, M — medium model, D-S — small data, D-M — medium data; FR — French, ES — Spanish, DE — German, CS — Czech, RU — Russian.

[6]These are "new" OOVs, since the original test set of Wikitext-2 already contains "old" OOVs marked as <unk>.

Model	PTB	WT-2	CS	DE	ES	FR	RU
AWD-LSTM-Word w/o emb. dropout	61.38	68.50	410	241	145	151	232
AWD-LSTM-MorphSum + RE + RW	61.17	66 .92	253	177	126	140	162

Table 7: Replacing LSTM with AWD-LSTM.

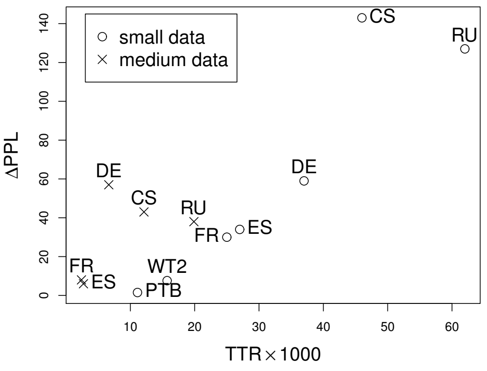

Figure 4: PPL improvement vs TTR. $\Delta\text{PPL} = \text{PPL}_{\text{Word+RE}} - \text{PPL}_{\text{MorphSum+RE+RW}}$.

6.6 Replacing LSTM with AWD-LSTM

The empirical perplexities in Table 2 are way above the current state-of-the-art on the same datasets (Melis et al., 2018). However, the approach of Melis et al. (2018) requires thousands of evaluations and is feasible for researchers who have access to hundreds of GPUs. Unfortunately, we do not have such access. Also, the authors do not disclose the optimal hyperparameters they found, and thus we could not reproduce their models. There is another state-of-the-art language model, AWD-LSTM (Merity et al., 2018), which has open-source code. We replaced this model's word embedding layer with the *MorphSum* subnetwork and fully reused morpheme embeddings and other weights of *MorphSum* at output. We refer to such modification as *AWD-LSTM-MorphSum + RE + RW*. We trained both models without fine-tuning (due to time constraints) and we did not use embedding dropout (section 4.3 of Merity et al. (2018)) in either model, as it is not obvious how embeddings should be dropped in the case of *AWD-LSTM-MorphSum*. The results of evaluation on the PTB, Wikitext-2, and non-English datasets are given in Table 7.

Although *AWD-LSTM-MorphSum* is on par with *AWD-LSTM-Word* on PTB and is slightly better on Wikitext-2, replacing plain word embeddings with the subword-aware model with appropriately reused parameters is crucial for non-English data. Notice that AWD-LSTM underperforms LSTM (used by us) on Czech dataset (cf. Table 6). We think that the hyperparameters of AWD-LSTM in Merity et al. (2018) are thoroughly tuned for PTB and Wikitext-2 and may poorly generalize to other datasets.

7 Conclusion

There is no single best way to reuse parameters in all subword-aware neural language models: the reusing method should be tailored to each type of subword unit and embedding model. However, instead of testing an exponential (w.r.t. sub-network depth) number of configurations, it is sufficient to check only those where weights are tied consecutively bottom-up.

Despite being similar, input and output embeddings solve different tasks. Thus, fully tying input and output embedding sub-networks in subword-aware neural language models is worse than letting them be slightly different. This raises the question whether the same is true for pure word-level models, and we defer its study to our future work.

One of our best configurations, a simple morpheme-aware model which sums morpheme embeddings and fully reuses the embedding sub-network, outperforms the competitive word-level language model while significantly reducing the number of trainable parameters. However, the performance gain diminishes with the increase of training set size.

Acknowledgements

We gratefully acknowledge the NVIDIA Corporation for their donation of the Titan X Pascal GPU used for this research. The work of Zhenisbek Assylbekov has been funded by the Committee of Science of the Ministry of Education and Science of the Republic of Kazakhstan, contract # 346/018-2018/33-28, IRN AP05133700. The authors would like to thank anonymous reviewers for their valuable feedback, and Dr. J. N. Washington for proofreading an early version of the paper.

References

Zhenisbek Assylbekov, Rustem Takhanov, Bagdat Myrzakhmetov, and Jonathan N. Washington. 2017. Syllable-aware neural language models: A failure to beat character-aware ones. In *Proc. of EMNLP*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2001. A neural probabilistic language model `http://www.iro.umontreal.ca/œlisa/pointeurs/nips00_lm.ps`.

Jan Botha and Phil Blunsom. 2014. Compositional morphology for word representations and language modelling. In *Proc. of ICML*.

Wenlin Chen, David Grangier, and Michael Auli. 2016. Strategies for training large vocabulary neural language models. In *Proc. of ACL*.

Ryan Cotterell and Hinrich Schütze. 2015. Morphological word-embeddings. In *Proc. of HLT-NAACL*.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Proc. of NIPS*.

Justin Garten, Kenji Sagae, Volkan Ustun, and Morteza Dehghani. 2015. Combining distributed vector representations for words. In *In Proc. of VS@HLT-NAACL*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2017. Tying word vectors and word classifiers: A loss framework for language modeling. In *Proc. of ICLR*.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On using very large target vocabulary for neural machine translation. In *Proc. of ACL-IJCNLP*.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410* .

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *Proc. of ICML*.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Proc. of AAAI*.

Matthieu Labeau and Alexandre Allauzen. 2017. Character and subword-based word representation for neural language modeling prediction. In *Proc. of SCLeM@EMNLP*.

Franklin Mark Liang. 1983. *Word Hy-phen-a-tion by Com-put-er*. Citeseer.

Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *Proc. of EMNLP*.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics* 19(2):313–330.

Gábor Melis, Chris Dyer, and Phil Blunsom. 2018. On the state of the art of evaluation in neural language models. In *Proc. of ICLR*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and optimizing lstm language models .

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *Proc. of ICLR*.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proc. of INTERSPEECH*.

Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, Stefan Kombrink, and Jan Cernocky. 2012. Subword language modeling with neural networks. *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)* .

Andriy Mnih and Geoffrey Hinton. 2007. Three new graphical models for statistical language modelling. In *Proc. of ICML*.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proc. of EACL*.

Siyu Qiu, Qing Cui, Jiang Bian, Bin Gao, and Tie-Yan Liu. 2014. Co-learning of word representations and morpheme representations. In *Proc. of COLING*.

Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Training very deep networks. In *Proc. of NIPS*.

Clara Vania and Adam Lopez. 2017. From characters to words to in between: Do we capture morphology? In *Proc. of ACL*.

Lyan Verwimp, Joris Pelemans, Patrick Wambacq, et al. 2017. Character-word lstm language models. In *Proc. of EACL*.

Sami Virpioja, Peter Smit, Stig-Arne Grönroos, Mikko Kurimo, et al. 2013. Morfessor 2.0: Python implementation and extensions for morfessor baseline .

Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proc. of the IEEE* 78(10):1550–1560.

Seunghak Yu, Nilesh Kulkarni, Haejun Lee, and Jihie Kim. 2017. Syllable-level neural language model for agglutinative language. In *Proc. of SCLeM@EMNLP*.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329* .

A Optimization

Training the models involves minimizing the negative log-likelihood over the corpus $w_{1:K}$:

$$-\sum_{k=1}^{T} \log \Pr(w_k|w_{1:k-1}) \longrightarrow \min,$$

by truncated BPTT (Werbos, 1990). We backpropagate for 35 time steps using stochastic gradient descent where the learning rate is initially set to

- 1.0 in small word-level models,
- 0.5 in small and medium *CharCNN*, medium *SylConcat* (SS, SS+RW) models,
- 0.7 in all other models,

and start decaying it with a constant rate after a certain epoch. This is 5 and 10 for the small word-level and all other networks respectively except *CharCNN*, for which it is 12. The decay rate is 0.9. The initial values for learning rates were tuned as follows: for each model we start with 1.0 and decrease it by 0.1 until there is convergence at the very first epoch. We use a batch size of 20. We train for 70 epochs. Parameters of the models are randomly initialized uniformly in $[-0.1, 0.1]$ and in $[-0.05, 0.05]$ for the small and medium networks, except the forget bias of the word-level LSTM, which is initialized to 1, and the transform bias of the highway layer, which is initialized to values around -2. For regularization we use a variant of variational dropout (Gal and Ghahramani, 2016) proposed by Inan et al. (2017). For PTB, the dropout rates are 0.3 and 0.5 for the small and medium models. For Wikitext-2, the dropout rates are 0.2 and 0.4 for the small and medium models. We clip the norm of the gradients (normalized by minibatch size) at 5.

For non-English small-sized data sets (Data-S) we use the same hyperparameters as for PTB. To speed up training on non-English medium-sized data (Data-M) we use a batch size of 100 and sampled softmax (Jean et al., 2015) with the number of samples equal to 20% of the vocabulary size (Chen et al., 2016).

	Data set	T	$\|\mathcal{W}\|$	$\|\mathcal{M}\|$
Small	French (FR)	1M	25K	6K
	Spanish (ES)	1M	27K	7K
	German (DE)	1M	37K	8K
	Czech (CS)	1M	46K	10K
	Russian (RU)	1M	62K	12K
Medium	French (FR)	57M	137K	26K
	Spanish (ES)	56M	152K	26K
	German (DE)	51M	339K	39K
	Czech (CS)	17M	206K	34K
	Russian (RU)	25M	497K	56K

Table 8: Non-English corpora statistics. T = number of tokens in training set; $|\mathcal{W}|$ = word vocabulary size; $|\mathcal{M}|$ = morph vocabulary size.

	Model	FR	ES	DE	CS	RU	
Small	Word + RE	5.6	6.1	8.0	10.0	13.4	Data-S
	MorphSum + RE + RW	2.1	2.3	2.5	2.8	3.2	
Medium	Word + RE	23.0	24.3	30.6	36.9	48.0	
	MorphSum + RE + RW	12.6	13.2	13.8	15.0	16.1	
Small	Word + RE	28.2	31.2	68.8	42.0	100.6	Data-M
	MorphSum + RE + RW	6.2	6.1	8.9	7.7	12.6	

Table 9: Model sizes in millions of trainable parameters.

B Non-English corpora statistics and model sizes

The non-English small and medium data comes from the 2013 ACL Workshop on Machine Translation[7] with pre-processing per Botha and Blunsom (2014). Corpora statistics is provided in Table 8.

Model sizes for *Word+RE* and *MorphSum+RE+RW*, which were evaluated on non-English data sets, are given in Table 9. *MorphSum+RE+RW* requires 45%–87% less parameters than *Word+RE*.

[7]`http://www.statmt.org/wmt13/translation-task.html`