

Patterns versus Characters in Subword-aware Neural Language Modeling

Rustem Takhanov and Zhenisbek Assylbekov

Nazarbayev University, Astana, Kazakhstan,
{rustem.takhanov, zhassylbekov}@nu.edu.kz

Abstract. Words in some natural languages can have a composite structure. Elements of this structure include the root (that could also be composite), prefixes and suffixes with which various nuances and relations to other words can be expressed. Thus, in order to build a proper word representation one must take into account its internal structure. From a corpus of texts we extract a set of frequent subwords and from the latter set we select patterns, i.e. subwords which encapsulate information on character n -gram regularities. The selection is made using the pattern-based Conditional Random Field model [23,19] with l_1 regularization. Further, for every word we construct a new sequence over an alphabet of patterns. The new alphabet's symbols confine a local statistical context stronger than the characters, therefore they allow better representations in \mathbb{R}^n and are better building blocks for word representation. In the task of subword-aware language modeling, pattern-based models outperform character-based analogues by 2-20 perplexity points. Also, a recurrent neural network in which a word is represented as a sum of embeddings of its patterns is on par with a competitive and significantly more sophisticated character-based convolutional architecture.

Keywords: subword-aware language modeling, pattern-based conditional random field, word representation, deep learning

1 Introduction

The goal of natural language modeling is, given a corpus of texts from a certain language, to build a probabilistic distribution over all possible sequences of words/sentences. Historically, first approaches to the problem [16,4] were highly interpretable, involving syntax and morphology, i.e. the internal structure of such models was of interest even to linguists. Nowadays the best performance is achieved by the so called recurrent neural network language models (RNNLM), which unfortunately lack the desired properties of interpretability.

For rich-resource languages the amount of training data, i.e. a corpus of texts, is bounded only by the computational power of the language modeling method. Due to this, most of RNNLM methods treat text as a sequence of token identifiers, where a token corresponds to either a word, or punctuation mark. Indeed, if any word appears in a text in various different contexts, a

method can learn high quality word representation without taking into account its morphology. This logic fails when a corpus of texts is not large enough, and the problem is aggravated for morphology-rich languages, such as, e.g., turkic or finno-ugric languages. Thus, the problem of word representation that would take into account an internal structure of a word becomes very actual — recent advances in language modeling are connected with treating words as sequences of characters or other subword units.

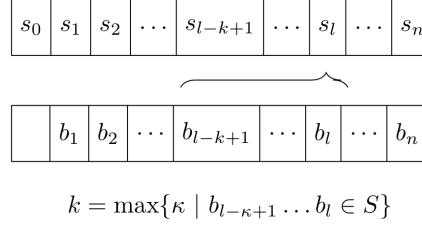
Much research has been done on character-level neural language modeling [15,6,11,9,10,20]. However, not much work exploits character n -grams that occur in a word. In [17] a word is represented using a character n -gram count vector, followed by a single nonlinear transformation to yield a low-dimensional embedding; the word embeddings are then fed into neural machine translation models. In [22] a very similar technique is used and an evaluation on three other tasks (word similarity, sentence similarity, and part-of-speech tagging) is performed; they demonstrate that their method outperforms more complex architectures based on character-level recurrent and convolutional neural networks. Probably closest to ours is an approach from [2] where a word representation is a sum of terms, each term corresponding to a certain n -gram that occurs in that word. One weakness of the mentioned approaches is that all possible n -grams that occur in a corpus of texts are present there in an *a priori* equal way, and a difference in their value for word representation is calculated in the process of learning. Whereas we in advance select a subset of n -grams that could potentially enrich word vectors by subword information. For this purpose we use the pattern-based Conditional Random Field with l_1 regularization.

Our approach also differs in the following aspects: we (i) replace each character by a new symbol which in some way concentrates an information on previous characters, (ii) experiment with several ways of combining subword embeddings to produce word embeddings, and (iii) evaluate our methods on a ubiquitous language modeling task.

2 A new alphabet for words

Throughout the paper, we will use the following notation: if \mathcal{X} is an alphabet, then \mathcal{X}^* denotes a set of words over \mathcal{X} ; for $\alpha, \beta \in \mathcal{X}^*$, $\alpha\beta$ denotes the concatenation of α and β ; by $*$ we denote an arbitrary word.

The key trick that we use in this paper is replacing a word $a_1a_2 \cdots a_k$ (that occurs in some context) over the initial alphabet \mathcal{A} with a word $s_1s_2 \cdots s_k$ over a new alphabet of states \mathcal{S} . Let us describe this substitution. We first define a finite state machine $(\mathcal{A}, \mathcal{S}, \delta, s_0)$, where s_0 is an initial state and $\delta : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is a state-transition function. If we are given a sentence $\alpha = b_1b_2 \cdots b_K$ such that every b_i is a character symbol from \mathcal{A} (it could be a punctuation mark, i.e. a symbol that marks a boundary between words) our state machine reads this sentence and produces a sequence of states: $s_0s_1 \cdots s_K$. In the latter sequence, every s_i corresponds to a state of our machine after reading a symbol b_i . Thus, every subsequence $b_ib_{i+1} \cdots b_j$ of the initial sentence α corresponds

**Fig. 1.** Finite-State Machine.

to a subsequence $s_i s_{i+1} \dots s_j$ where $1 \leq i \leq j \leq K$. Therefore, if $b_i b_{i+1} \dots b_j$ corresponds to a word in a sentence α , then we will substitute it with $s_i s_{i+1} \dots s_j$.

Thus, given such a finite state machine, every word of a sentence can be rewritten over another alphabet \mathcal{S} . Let us describe now our finite state machine.

Suppose that after an analysis of a training set, i.e. of a corpus of texts from our language \mathcal{L} , we extract a certain finite set of sequences $\Pi_0 \subseteq \mathcal{A}^*$ that we assume not only to be frequent, but in some way statistically characterising our language. A specific way of choosing Π_0 will be given in the following subsection. Any element $\pi \in \Pi_0$ we call a *pattern*. Any such set defines a set of states $\mathcal{S} = \{\beta \mid \exists \pi = \beta*\}$, which is, in fact, a set of all prefixes of patterns. We assume that an empty word ε is also in \mathcal{S} and define $s_0 = \varepsilon$.

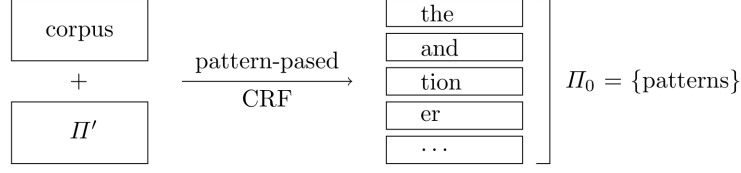
Now we have to define a state-transition function δ . Our idea is to construct it in such a way that after reading the first l symbols of the sentence $b_1 b_2 \dots b_l$ the machine should be in a state $s_l \in \mathcal{S}$ where s_l is the longest word from \mathcal{S} for which $b_1 b_2 \dots b_l = *s_l$ (Figure 1). The latter description induces the following definition: for any $\alpha \in \mathcal{S}$ and $a \in \mathcal{A}$, $\delta(\alpha, a)$ is the longest word $\beta \in \mathcal{S}$ for which $\alpha a = *\beta$.

Patterns

In this subsection we will describe how we extract a set of patterns Π_0 from a corpus of texts (Figure 2). By a corpus of texts we understand a training set $T = \{\alpha_1, \dots, \alpha_L\} \subseteq \mathcal{A}^*$ where α_i is a sentence from our language \mathcal{L} .

First we extract from our training set T a set of patterns Π' based on the following simple procedure: we fix in advance a threshold f and put to T only those words $\alpha \in \mathcal{A}^*$ that occur in T in more than f places. Then we apply a reduction procedure, i.e. if a) α is a subword of β , b) α and β always occur together in T , then we delete α from Π' . A pattern-based conditional random field model for our language is the following probability distribution over \mathcal{A}^* [23,19]:

$$\Pr(b_1 \dots b_K) = A \cdot e^{-E(b_1 \dots b_K)},$$

**Fig. 2.** Pattern mining.

where $E(b_1 \dots b_K) = \sum_{\alpha \in \Pi'} \sum_{i < j: b_i \dots b_j = \alpha} c^\alpha$, and c^α , $\alpha \in \Pi'$, are parameters to be learned from T .

The learning is done by the minimization of the negative log-likelihood with L_1 -regularization:

$$- \sum_{i=1}^L \log \Pr(\alpha_i) + C \sum_{\alpha \in \Pi'} |c^\alpha|. \quad (1)$$

The latter function is convex, an efficient computation of its value and gradient is described in [19]. For the optimization we used the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method written by Jorge Nocedal. Via the parameter C one can manage the number of patterns $\alpha \in \Pi'$ for which $c^\alpha \neq 0$. Finally, we define $\Pi_0 = \{\alpha \in \Pi' | c^\alpha \neq 0\}$.

3 Subword-aware neural language model

In what follows, both regular characters and patterns are referred to as *subwords*. The overall architecture of the subword-aware neural language model is displayed in Figure 3.

It consists of three main parts: (i) subword-based word embedding model, (ii) word-level recurrent neural network language model (RNNLM), and (iii) softmax layer. Below we describe each part in more detail.

Subword-based word embeddings: A word $w \in \mathcal{W}$ (in a sentence) is defined by the sequence of its subwords $s_1 \dots s_{n_w} \in \mathcal{X}^*$ ($\mathcal{X} = \mathcal{A}$ in the case of character-based representation, and $\mathcal{X} = \mathcal{S}$ in our pattern-based approach), and each state is embedded into $d_{\mathcal{X}}$ -dimensional space via an embedding matrix $\mathbf{E}_{\mathcal{X}}^{\text{in}} \in \mathbb{R}^{|\mathcal{X}| \times d_{\mathcal{X}}}$ to obtain a sequence of state vectors:

$$\mathbf{s}_1, \dots, \mathbf{s}_{n_w}. \quad (2)$$

Then we try three different methods to get an embedding of the word w :

– **Concat:** A simple concatenation of state vectors (2) into a single word vector:

$$\mathbf{w} = [\mathbf{s}_1; \mathbf{s}_2; \dots; \mathbf{s}_{n_w}; \underbrace{\mathbf{0}; \mathbf{0}; \dots; \mathbf{0}}_{n-n_w}].$$

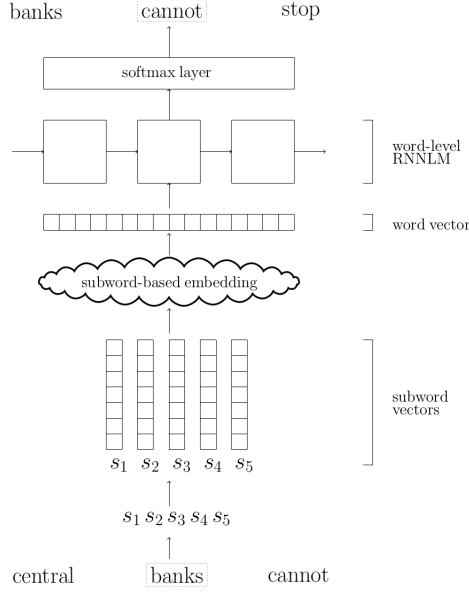


Fig. 3. Subword-aware language model.

We either truncate (if w consists of more than n symbols) or zero-pad \mathbf{w} so that all word vectors have the same length $n \cdot d_{\mathcal{X}}$ to allow batch processing. This approach is motivated by a desire to keep all the information regarding subwords, including the order in which they appear in the word.

- **Sum:** A summation of subword vectors:

$$\mathbf{w} = \sum_{t=1}^{n_w} \mathbf{s}_t. \quad (3)$$

This approach was used by [3] to combine a word and its morpheme embeddings into a single word vector.

- **CNN:** A convolutional model of [9]:

$$\mathbf{w} = \text{CNN}(\mathbf{s}_1, \dots, \mathbf{s}_{n_w}).$$

This method has already demonstrated excellent performance for character-level inputs, therefore we decided to apply it to patterns as well.

To model interactions between subwords, we feed the resulting word embedding \mathbf{w} into a stack of two highway layers [18] with dimensionality d_{HW} per layer. In cases when dimensionality of \mathbf{w} does not match d_{HW} , we project it into $\mathbb{R}^{d_{\text{HW}}}$. **Word-level RNNLM:** Once we have embeddings $\mathbf{w}_{1:k}$ for a sequence of words $w_{1:k}$, we can use a word-level RNN language model to produce a sequence of states $\mathbf{h}_{1:k} \in \mathbb{R}^{d_{\text{LM}}}$ according to

$$\mathbf{h}_t = \text{RNNCell}(\mathbf{w}_t, \mathbf{h}_{t-1}), \quad \mathbf{h}_0 = \mathbf{0}.$$

There is a big variety of RNN cells to choose from. The most advanced recurrent neural architectures, at the time of this writing, are RHN [25] and NAS [26]. However, to make our results directly comparable to the previous work of [9] on character-level language modeling we select a more conventional architecture – a stack of two LSTM cells [8].

Softmax: The last state \mathbf{h}_k from (4) is further used to predict the next word w_{k+1} according to the probability distribution

$$\Pr(w_{k+1}|w_{1:k}) = \text{softmax}(\mathbf{h}_k \mathbf{W} + \mathbf{b}), \quad (4)$$

where $\mathbf{W} \in \mathbb{R}^{d_{\text{LM}} \times |\mathcal{W}|}$, $\mathbf{b} \in \mathbb{R}^{|\mathcal{W}|}$, and d_{LM} is a hidden layer size of the RNN.

4 Experimental Setup

Data sets: All models are trained and evaluated on the English PTB data set [12] utilizing the standard training (0-20), validation (21-22), and test (23-24) splits along with pre-processing by [14]. Since the PTB is criticized for being small nowadays, we also provide an evaluation on the WikiText-2 data set [13], which is approximately two times larger than PTB in size and three times larger in vocabulary. We do not append any additional symbols at the end of each line in WikiText-2, but remove spaces between equality signs in the sequences “=” and “= = =”, which occur in section titles.

Hyperparameters: The regularization parameter C from (1) is set to 1600, which results in 883 unique patterns ($|\Pi_0| = 883$, $|\mathcal{S}| = 890$) for the PTB data set (cf. 48 plain characters) and 1440 unique patterns ($|\Pi_0| = 1440$, $|\mathcal{S}| = 1471$) for the WikiText-2 data set (cf. 281 plain characters). We set the threshold value f to 300 on the PTB and to 700 on the WikiText-2. We experiment with two configurations for the state size d_{LM} of the word-level RNNLM: 300 (small models) and 650 (medium-sized models). Specification of other hyperparameters is given below.

Concat: $d_{\mathcal{A}} = 15$ (for characters), and $d_{\mathcal{S}} = 30$ (for patterns). We give higher dimensionality to patterns as their amount significantly exceeds the amount of characters. n is set to the 95th percentile of word lengths, i.e. 95% of all words have not more than n characters¹. We do not set $n = \max_{w \in \mathcal{W}} n_w$, as this would result in excessive zero-padding. $d_{\text{HW}} = d_{\text{LM}}$.

Sum: $d_{\mathcal{X}} = d_{\text{HW}} = d_{\text{LM}} \in \{300, 650\}$ for both characters and patterns. We give higher dimensionality to subword vectors here (compared to other models) since the resulting word vector will have the same size as subword vectors (see (3)).

CNN: In character-based models we choose the same values for hyperparameters as in the work of [9]. For pattern-based models we choose: $d_{\mathcal{S}} = 50$ and $d_{\mathcal{S}} = 100$ for small and medium-sized models; filter widths are [1, 2, 3, 4, 5, 6] and [1, 2, 3, 4, 5, 6, 7] for small and medium-sized models; the corresponding

¹ word length in characters and in patterns is the same.

depths (number of features per width) are [100, 50, 75, 100, 100, 100] and [100, 100, 150, 200, 200, 200, 200]. $d_{\text{HW}} = \sum \text{depths} \in \{525, 1150\}$.

Optimization is done similarly to [24,9,5]. Training the models involves minimizing the negative log-likelihood over the corpus $w_{1:K}$:

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

which is typically done by truncated BPTT [21,6]. We backpropagate for 35 time steps using stochastic gradient descent where the learning rate is initially set to 0.7 and halved if the perplexity does not decrease on the validation set after an epoch. We use a batch size of 20. We train for 65 epochs, picking the best performing model on the validation set. Parameters of the models are randomly initialized uniformly in $[-0.05, 0.05]$, except the forget bias of the word-level LSTM, which is initialized to 1, and the transform bias of the highway, which is initialized to values near -2 . For regularization we use variational dropout [5] with dropout rates for small/medium Concat, Sum/medium CNN models as follows: 0.1/0.15/0.2 for the embedding layer, 0.2/0.3/0.35 for the input to the gates, 0.1/0.15/0.2 for the hidden units, and 0.2/0.3/0.35 for the output activations. We clip the norm of the gradients (normalized by minibatch size) at 5.

5 Results

The results of evaluation on PTB and WikiText-2 are reported in Tables 1 and 2 correspondingly. As one can see, models which process patterns consistently outperform those which use characters under small parameter budgets. However, the difference in performance is less pronounced when we allow more parameters.

Table 1. Results on the PTB for small (left) and medium-sized models.

Model	Characters		Patterns		Model	Characters		Patterns	
	Size	PPL	Size	PPL		Size	PPL	Size	PPL
Concat	5M	119.2	5M	99.6	Concat	15M	91.5	15.8M	83.6
Sum	5M	108.2	5M	87.4	Sum	15M	91.5	15.5M	82.1
CNN	6M	87.3	6M	84.8	CNN	20M	79.6	20.5M	77.2

Table 2. Results on WikiText-2 for small (left) and medium-sized models.

Model	Characters		Patterns		Model	Characters		Patterns	
	Size	PPL	Size	PPL		Size	PPL	Size	PPL
Concat	11.9M	138.2	12.1M	114.2	Concat	30.2M	115.9	30.8M	99.0
Sum	11.9M	124.0	12.3M	101.9	Sum	30.3M	106.7	31.1M	94.9
CNN	12.9M	105.2	13.0M	102.8	CNN	34.5M	97.38	35.7M	94.2

Also, it is clearly seen that patterns are more beneficial for simple models, such as Concat and Sum, but have less effect on the CNN model, which shrinks

the gap between characters and patterns. This is quite natural as patterns carry some information on character n -grams and, hence, can be considered as “discrete convolutions”, which makes CNN over patterns not as efficient as CNN over regular characters. However, we notice that in all cases a simple sum of pattern embeddings (Pat-Sum) is on par with a more sophisticated convolution over character embeddings (Char-CNN). Faster² training of the Pat-Sum compared to the Char-CNN makes the patterns even more advantageous.

Why does Pat-Sum perform equally well as Char-CNN? As was described in Section 3 word embeddings are processed by the two highway layers before they are fed into the RNNLM. Highway is a weighted average between nonlinear and identity transformations of the incoming word embedding:

$$\mathbf{w} \mapsto \mathbf{t} \odot \sigma(\mathbf{w}\mathbf{A} + \mathbf{b}) + (\mathbf{1} - \mathbf{t}) \odot \mathbf{w},$$

where \mathbf{t} , \mathbf{A} and \mathbf{b} are trainable parameters, $\sigma(\cdot)$ is a non-linear activation, $\mathbf{1}$ is a vector whose all components are 1 and \odot is an operation of component-wise multiplication. The ideal input for the highway is the one that does not need to undergo a nonlinear transformation, i.e the highway will then be close to an identity operator, and hence in the ideal case we shall have $\mathbf{t} = \mathbf{0}$. But if \mathbf{w} is rather “raw”, then the highway should prepare it for the RNN (resulting in $\mathbf{t} \neq \mathbf{0}$). Such extra nonlinearity can be measured by the closeness of \mathbf{t} to $\mathbf{1}$. We hypothesize that the reason why Pat-Sum performs well is that the sum of pattern embeddings is *already* a good word representation. Hence the highway in Pat-Sum does less nonlinear work than in Char-CNN: In Pat-Sum it is almost an identical transformation, and such a simple highway is well-trained according to [7]. To validate our hypothesis we compare the distributions of the transform

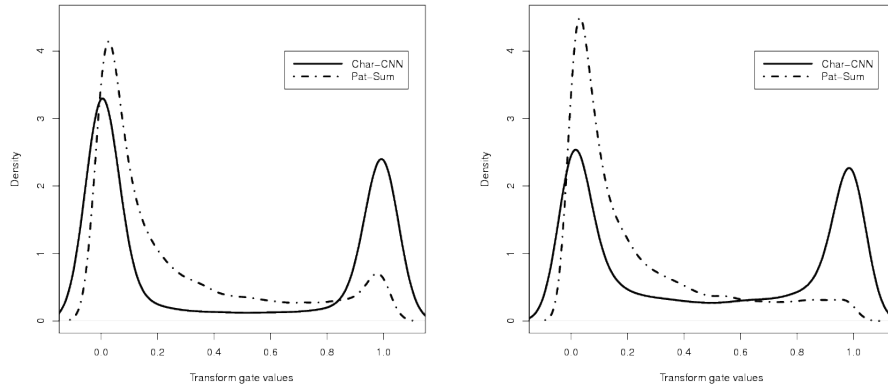


Fig. 4. Kernel density estimations of the transform gate values of the first (left) and second highway layers in Char-CNN and Pat-Sum.

gate \mathbf{t} values from both highway layers of Pat-Sum and Char-CNN. The den-

² around 1.2x speedup on NVIDIA Titan X (Pascal)

sity plots in Fig. 4 support our hypothesis: Pat-Sum does not utilize much of nonlinearity in the highway layers, while Char-CNN heavily relies on it.

Source code: All models were implemented in TensorFlow [1] and the source code for Pat-Sum is available at <https://github.com/zh3nis/pat-sum>.

6 Conclusion

Regular characters are rather uninformative when their embeddings are concatenated or summed to produce word vectors, but patterns, on the contrary, carry enough information to make these methods work significantly better. Convolutions over subword embeddings do capture n -gram regularities and, therefore, make the difference between characters and patterns less noticeable. It is noteworthy, that a simple and fast sum of pattern embeddings is on par with more sophisticated and slower convolutions over characters embeddings.

7 Acknowledgments

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)
2. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. arXiv preprint arXiv:1607.04606 (2016)
3. Botha, J., Blunsom, P.: Compositional morphology for word representations and language modelling. In: Proceedings of the 31st International Conference on Machine Learning (ICML-14). (2014) 1899–1907
4. Chomsky, N.: Three models for the description of language. IRE Transactions on information theory **2**(3) (1956) 113–124
5. Gal, Y., Ghahramani, Z.: A theoretically grounded application of dropout in recurrent neural networks. In: Advances in Neural Information Processing Systems. (2016) 1019–1027
6. Graves, A.: Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850 (2013)
7. Hardt, M., Ma, T.: Identity matters in deep learning. arXiv preprint arXiv:1611.04231 (2016)
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8) (1997) 1735–1780
9. Kim, Y., Jernite, Y., Sontag, D., Rush, A.M.: Character-aware neural language models. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI Press (2016) 2741–2749
10. Lankinen, M., Heikinheimo, H., Takala, P., Raiko, T., Karhunen, J.: A character-word compositional neural language model for finnish. arXiv preprint arXiv:1612.03266 (2016)

11. Ling, W., Dyer, C., Black, A.W., Trancoso, I., Fernandez, R., Amir, S., Marujo, L., Luis, T.: Finding function in form: Compositional character models for open vocabulary word representation. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal, Association for Computational Linguistics (September 2015) 1520–1530
12. Marcus, M.P., Marcinkiewicz, M.A., Santorini, B.: Building a large annotated corpus of english: The penn treebank. *Computational linguistics* **19**(2) (1993) 313–330
13. Merity, S., Xiong, C., Bradbury, J., Socher, R.: Pointer sentinel mixture models. In: Proceedings of ICLR 2017. (2017)
14. Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., Khudanpur, S.: Recurrent neural network based language model. In: Interspeech. Volume 2. (2010) 3
15. Mikolov, T., Sutskever, I., Deoras, A., Le, H.S., Kombrink, S., Cernocky, J.: Sub-word language modeling with neural networks. preprint ([http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf](http://www.fit.vutbr.cz/imikolov/rnnlm/char.pdf)) (2012)
16. Shannon, C.E., Weaver, W.: A mathematical theory of communication. (1963)
17. Sperr, H., Niehues, J., Waibel, A.: Letter n-gram-based input encoding for continuous space language models. In: Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality. (2013) 30–39
18. Srivastava, R.K., Greff, K., Schmidhuber, J.: Training very deep networks. In: Advances in neural information processing systems. (2015) 2377–2385
19. Takhanov, R., Kolmogorov, V.: Inference algorithms for pattern-based crfs on sequence data. In: ICML (3). (2013) 145–153
20. Verwimp, L., Pelemans, J., Wambacq, P., et al.: Character-word lstm language models. In: Proceedings of EACL 2017. (2017)
21. Werbos, P.J.: Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE* **78**(10) (1990) 1550–1560
22. Wieting, J., Bansal, M., Gimpel, K., Livescu, K.: Charagram: Embedding words and sentences via character n-grams. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1–4, 2016. (2016) 1504–1515
23. Ye, N., Lee, W.S., Chieu, H.L., Wu, D.: Conditional random fields with high-order features for sequence labeling. In: Advances in Neural Information Processing Systems. (2009) 2196–2204
24. Zaremba, W., Sutskever, I., Vinyals, O.: Recurrent neural network regularization. arXiv preprint arXiv:1409.2329 (2014)
25. Zilly, J.G., Srivastava, R.K., Koutník, J., Schmidhuber, J.: Recurrent highway networks. arXiv preprint arXiv:1607.03474 (2016)
26. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: Proceedings of ICLR 2017. (2017)