CAPITALIZATION NORMALIZATION FOR LANGUAGE MODELING WITH AN ACCURATE AND EFFICIENT HIERARCHICAL RNN MODEL

Hao Zhang, You-Chi Cheng, Shankar Kumar, W. Ronny Huang, Mingqing Chen, Rajiv Mathews

Google Research
haozhang@google.com

ABSTRACT

Capitalization normalization (truecasing) is the task of restoring the correct case (uppercase or lowercase) of noisy text. We propose a fast, accurate and compact two-level hierarchical word-and-character-based recurrent neural network model. We use the truecaser to normalize user-generated text in a Federated Learning framework for language modeling. A case-aware language model trained on this normalized text achieves the same perplexity as a model trained on text with gold capitalization. In a real user A/B experiment, we demonstrate that the improvement translates to reduced prediction error rates in a virtual keyboard application. Similarly, in an ASR language model fusion experiment, we show reduction in uppercase character error rate and word error rate.

Index Terms— Text normalization, capitalization, truecasing, language modeling.

1. INTRODUCTION

The vast amount of online text powers language models for speech recognition, typing suggestions and many other language generation tasks. However user-generated texts, especially those from mobile applications such as Twitter Tweets [1], often violate the grammatical rules of casing in English and other western languages [2]. The process of restoring the proper case, often known as **tRuEcasIng** [3], provides a factorized solution with a dedicated model for case normalization. Such a model can be applied on noisy text before case-aware language model training or on model output after case-agnostic language model inference. Concretely, it has been used for restoring case and improving the recognition performance for ASR (*post-processing*, [4, 5, 6]) as well as case normalization of user text prior to language model (LM) training (*pre-processing*, [7]). [7] employs Federated Learning [8], a privacy-preserving learning paradigm, on distributed devices. The need for normalizing text on a variety of mobile devices makes it an imperative to develop a fast, accurate and compact truecaser.

From a modeling perspective, prior work can be grouped into word-based and character-based approaches. Most of the earlier works are word-based [3, 9]. In the word-based view, the task is to classify each word in the input into one of the few classes [3]: all lowercase (`LC`), first letter uppercase (`UC`), all letters uppercase (`CA`), and mixed case (`MC`). The main drawback of the word-based approach is that it does not generalize well to unseen words and mixed case words. The development of character-based neural network modeling techniques [10] led to the introduction of character-based truecasers [11, 12]. In the character-based view, the task is to classify each character in the input into one of the two classes: `U` and `L`, for uppercase and lowercase respectively. The shortcoming of character-based models is inefficiency. As word-level features in some form are important for classification, character-based models need to be deep enough to capture word-level features, which exacerbates their slowness.

We view truecasing as a special case of text normalization [13] with similar trade-offs for accuracy and efficiency. Thus, we classify input words into one of two classes: `SELF` and `OTHER`. Words labeled `SELF` will be copied as-is to the output. Words labeled `OTHER` will be fed to a sub-level character-based neural network along with the surrounding context, which is encoded bidirectionally. The task of the sub-network is to perform a non-trivial transduction such as: `iphone` → `iPhone`, `mcdonald's` → `McDonald's` and `hewlett-packard` → `Hewlett-Packard`.

The two-level hierarchical network is fast while also being accurate. We report the model's intrinsic accuracy, speed, and size on a curated Wikipedia data set in Section 4.1. Next, we focus on case-aware language modeling on noisy text. In Section 4.2, we show that applying the model in preprocessing can reduce a cased language model's perplexity. Furthermore, in Section 4.3, we show that when the cased language model trained in this fashion is applied in a noisy channel virtual keyboard, it leads to reduction in word prediction errors in an A/B test on real users. Finally, in section 4.4, we show similar improvement in ASR language model fusion.

2. RELATED WORK

Word-based truecasing has been the dominant approach since the introduction of the task by [3]. Word-based models can be further categorized into generative models such as HMMs [3, 4, 5, 1] and discriminative models such as Maximum-Entropy

Markov Models [9], Conditional Random Fields [14], and most recently Transformer neural network models [15, 16, 6]. Word-based models need to refine the class of mixed case words because there is a combinatorial number of possibilities of case mixing for a word (e.g., LaTeX). [3, 9] suggested using either all of the forms or the most frequent form of mixed case words in the training data. The large-scale finite state transducer (FST) models in [5] used all known forms of mixed case words to build the "capitalization" FST. [14] used a heuristic **GEN** function to enumerate a superset of all forms seen in the training data. But others [15, 16] have chosen to simplify the problem by mapping mixed case words to first letter uppercase words. [6] only evaluated word class F1, without refining the class of MC.

Character-based models have been explored largely after the dawn of modern neural network models. [11] first introduced character-based LSTM for this task and completely solved the mixed case word problem. Recently, [2] compared character-based n-gram (n up to 15) language models with the character LSTM of [11]. [12] advanced the state of the art with a character-based CNN-LSTM-CRF model.

Text normalization is the process of transforming text into a canonical form. Examples of text normalization include but are not limited to written-to-spoken text normalization for speech synthesis [13], spoken-to-written text normalization for speech recognition [17], social media text normalization [18], and historical text normalization [19]. Truecasing is a problem that appears in both spoken-to-written and social media text normalization.

3. FORMULATION AND MODEL ARCHITECTURE

The input is a sequence of all lowercase words $\vec{\mathrm{X}} = (x_1, \ldots, x_l)$. The output is a sequence of words with proper casing $\vec{\mathrm{Y}} = (y_1, \ldots, y_l)$. We introduce a latent sequence of class labels $\vec{\mathrm{C}} = (c_1, \ldots, c_l)$, where $c_i \in \{\mathtt{S{=}SELF}, \mathtt{O{=}OTHER}\}$. We use the notation x_i^j and y_i^j to represent the j-th character within the i-th word.

The model is trained to predict the probability:

$$\mathrm{P}(\vec{\mathrm{Y}}|\vec{\mathrm{X}}) = \sum_{\vec{\mathrm{C}}} \mathrm{P}(\vec{\mathrm{Y}}|\vec{\mathrm{X}}, \vec{\mathrm{C}}) \cdot \mathrm{P}(\vec{\mathrm{C}}|\vec{\mathrm{X}}), \tag{1}$$

where $\mathrm{P}(\vec{\mathrm{C}}|\vec{\mathrm{X}})$ is a word-level model that predicts if a word should stay all lowercase (SELF) or change to a different case form (OTHER) taking into account label dependencies between $c_1, \ldots, c_{i-1}$ and c_i.

$$\mathrm{P}(\vec{\mathrm{C}}|\vec{\mathrm{X}}) = \prod_{i=1}^{l} \mathrm{P}(c_i|c_1, \ldots, c_{i-1}, \vec{\mathrm{X}}) \tag{2}$$

The label sequence $\vec{\mathrm{C}}$ works as a gating mechanism,

$$\mathrm{P}(\vec{\mathrm{Y}}|\vec{\mathrm{X}}, \vec{\mathrm{C}}) = \prod_{i=1}^{l} \delta(c_i, \mathtt{O})\mathrm{P}(y_i|\mathrm{X}) + \delta(c_i, \mathtt{S})\delta(x_i, y_i), \tag{3}$$

where $\mathrm{P}(y_i|\mathrm{X})$ is a character-level model that predicts each output character within a word, assuming dependency between characters within a word: $y_i^1, \ldots, y_i^{j-1}$ and y_i^j, but no cross-word dependency between $y_1, \ldots, y_{i-1}$ and y_i, and S and O denote SELF and OTHER respectively.

$$\mathrm{P}(y_i|\mathrm{X}) = \prod_{j=1}^{j=|x_i|} \mathrm{P}(y_i^j|y_i^1, \ldots, y_i^{j-1}, \vec{\mathrm{X}}) \tag{4}$$

Given that $\delta(c_i, \mathtt{S}) \equiv \delta(x_i, y_i)$, we can derive the log likelihood of Equation 1 as:

$$\log(\mathrm{P}(\vec{\mathrm{Y}}|\vec{\mathrm{X}})) = \sum_{i=1}^{l} \delta(c_i, \mathtt{O}) \log(\mathrm{P}(y_i|\mathrm{X})) + \log(\mathrm{P}(\vec{\mathrm{C}}|\vec{\mathrm{X}})) \tag{5}$$

Equations 2 and 4 can be modeled as sequence-to-sequence (seq2seq) problems. Unlike the general machine translation problem with unequal number of input and output tokens which requires a soft attention mechanism [20], both of our seq2seq problems can assume hard alignment between the output label at each time step and the input symbol at the same time step (c_i is aligned to x_i, y_i^j is aligned to x_i^j).

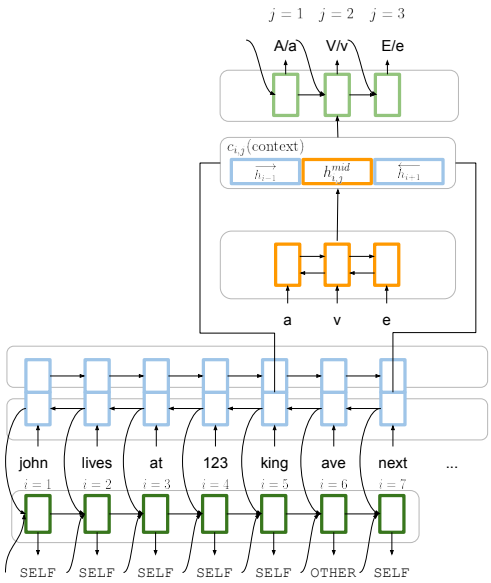

Fig. 1. Hierarchical RNN architecture.

We follow the multi-task recurrent neural network architecture of [13]. Figure 1 displays the rolled-out network of the two-level hierarchical RNN on a typical input sentence. The key difference with [13] is that in our setting, the models at both levels perform sequence tagging whereas their second-level model is a full-fledged sequence-to-sequence model with soft attention [20].

The exact inference requires searching over all possible $\vec{\mathrm{C}}$ and $\vec{\mathrm{Y}}$ which is infeasible for RNNs. But we can do beam

search for both Equation 2 and Equation 4 and use either the entire beam of hypotheses of $\vec{C}$ or simply the best one in Equation 3. In our experiments, using the entire beam only helped slightly.

4. EXPERIMENTS

Our models as depicted in Figure 1 have a pair of forward and backward RNNs over words to encode sentence context. Words are first encoded based on subword features, namely character n-grams with n up to 3 [13]. In the example shown in Figure 1, the character n-grams representing the word `ave` include `a`, `v`, `e`, `<s>a`, `av`, `ve`, `e<s>`, `<s>av`, `ave`, and `ve<s>`, where `<s>` is the placeholder symbol for the left and right boundaries. Each word is represented as the summation of the embeddings of the hashed character n-grams.

We train a large teacher model and then apply sequence distillation [21] to produce a compact student model. The hyper-parameters of both models are listed in Table 1. We use the same hyper-parameters for the word- and character-level sub-models.

	teacher (large)	student (small)
input embedding size	512	128
output embedding size	512	128
# of forward enc. layers	2	1
# of backward enc. layers	2	1
# of dec. layers	2	1
# of enc. GRU cells	512	128
# of dec. GRU cells	512	128
max char n-gram order	3	3
buckets of char n-grams	5000	5000
beam size	2	2

Table 1. Network hyper-parameters.

4.1. Accuracy, Speed, and Model Size Comparison

In this section, we report performance of models trained on a large data set of digitized books and newswire articles containing 1.5 billion sentences. We use 90% of the data for training and sample a subset from the remaining 10% for validation. We use a human-curated data set of 1200 sentences[1] from Wikipedia edit history [22] for testing.

All models take lowercase tokenized sentences as input and output the same tokens with predicted casing. The predicted tokens are compared against references. Following [11], we use non-lowercase (NL) F1 as the evaluation metric.

$$\text{NL Precision} = \frac{\text{\# of correct NL predictions}}{\text{\# of NL predictions}}$$

$$\text{NL Recall} = \frac{\text{\# of correct NL predictions}}{\text{\# of NL references}}$$

[1] https://github.com/google-research-datasets/wikipedia-intrinsic-capitalization

NL F1 is the harmonic mean of NL Precision and Recall. For brevity, we use precision, recall and F1 without qualification.

We present three groups of results in Table 2. The FST group represents word-based HMM models with a wide coverage of mixed case words. The character-based RNN group is our implementation of [11] with various hyper-parameters. The third group is our hierarchical RNNs.

We report CPU speed relative to the large FST model. We use batch size of 1 at inference time for all models. All models are implemented in TensorFlow and quantized after training. We compare the model sizes in terms of the total number of parameters.

FST models have high precision but low recall, showing the coverage problem of word-based models. The one-layer uni-directional character-based RNN scores low in both precision and recall. Even a single-layer bidirectional character-based RNN is already 4 times slower than a single-layer word- and character-based hierarchical RNN. Using two layers in both the encoder and the decoder, a purely character-based model is nearly 20 times slower than a hierarchical model with about the same F1 score.

4.2. Case-aware Language Models

In this section, we study the effect of capitalization for case-aware language models. We use the 1B Word Benchmark data set (LM1B) introduced by [23]. The training and evaluation sections consist of 0.8B words (30.4M sentences) and 153K words (6075 sentences) respectively. The vocabulary consists of 800K words. To simulate the scenario when the training data has noisy capitalization, we noisify the training data by randomly lowercasing either 25% or 50% of the capitalized words. The baseline system does not perform any capitalization normalization. Two systems to contrast with the baseline apply the FST capitalizer and the RNN capitalizer respectively. Without loss of generality, we use a moderately large (2-layer) sub-word (16k word pieces, 2048 hidden units) LSTM LM as the language model architecture [24].

Table 3 shows the word-level perplexities of the baseline (noisy capitalization), the two capitalization normalizers (predicted capitalization), and the oracle (ground-truth capitalization). We observe that the RNN capitalizer yields a similar perplexity as the oracle while outperforming the FST capitalizer by a small margin.

4.3. Case-aware Language Models in Virtual Keyboard Applications

Virtual keyboards are indispensable for text input on the popular touchscreen devices. As spatial input signals of tap and glide trails are often noisy, decoding the intended input can be achieved with a noisy channel model composed of a spatial model and a language model [25]. The quality of language models directly impacts the prediction accuracy of virtual keyboards which is reflected in features such as auto-corrections,

System		*Precision*	*Recall*	*F1*	*Speed*	*# of params*
5-gram FST	unpruned, unoptimized	91.64	43.55	59.04	1.0x	10M
	pruned, optimized	**91.88**	41.19	56.88	88.0x	1M
char. RNN	small, 1-layer uni-, dec.	69.11	22.86	34.35	0.7x	230K
	small, 1-layer bi-, enc.&dec.	86.12	75.07	80.22	0.5x	400K
	large, 2-layer bi-, enc.&dec.	87.06	78.09	82.33	0.1x	8.4M
hier. RNN, student	small, 1-layer bi-, enc.&dec. $\times 2$	86.95	79.81	83.23	2.2x	1.3M
hier. RNN, teacher	large, 2-layer bi-, enc.&dec. $\times 2$	88.01	**82.60**	**85.22**	0.3x	19.2M

Table 2. Large-scale comparison across model types on intrinsically capitalized Wikipedia edit history data set.

Capitalization Model	*Perplexity*
50% corrupt	59.41
25% corrupt	54.68
5-gram FST	51.74
hier. RNN	**51.60**
oracle	51.61

Table 3. Perplexities of RNN language models on LM1B using different capitalization normalization methods.

Model	*WMR*	*RAC*
5-gram FST	5.81%	2.91%
hier. RNN	5.78%	2.87%
Rel. Reduction	**[-0.92,-0.11]%**	**[-2.21, -0.69]%**

Table 4. Virtual keyboard A/B experiment results. *WMR* is the fraction of words modified or retyped. *RAC* is the auto-correction rejection rate. The last row shows the 95% confidence interval of the relative reductions.

word completions, and next word predictions. In this section, we report the results of an A/B experiment on real users. The two systems under comparison differ only in the cased language models they use in decoding. The baseline uses the pruned FST model for capitalization normalization. The new system uses the hierarchical RNN model for capitalization normalization. The normalization step is done in a distributed fashion on users' devices before privacy-preserving Federated Learning of neural language models [7] starts. The two resulting LMs have exactly the same architecture [7] and the same number of parameters. Both are trained for the same number of rounds which amount to billions of tokens.

Table 4 shows the results of the A/B test using metrics indicative of the prediction accuracy of the virtual keyboard. The test covers a time span long enough to accumulate statistics over more than 1 billion typed words. We observe reductions of word modification rate (WMR) and auto-correction rejection rate (RAC) in the virtual keyboard usage statistics. Even though references are impossible to obtain in the A/B test, these metrics approximate WER (Word Error Rate) with regard to the intended user input. The reductions are statistically significant as shown by the 95% confidence intervals. We also observe a small but non-significant reduction of Out-of-vocabulary (OOV) rate by 0.4%. The OOV rate changes because the vocabulary size is identical in both LMs. Capitalization normalization causes the frequency ranking of word types to change.

4.4. Case-aware Language Models in Speech Recognition

Our speech recognizer is a streaming Conformer acoustic model [26] integrated with a Conformer language model [27] via HAT shallow fusion [28], and we measure performance on a $\sim$10k sample of Google voice search traffic with natively capitalized reference transcripts. When the acoustic model is fixed and the output tokens are cased, we show that improving capitalization normalization of the language model training data leads to reduction of upper-case error rate (UER), the character error rate of each capitalized character in either the predicted or reference transcript (Table 5). Importantly, the overall case-insensitive WER is also reduced, likely due to the language model inducing more favorable beam search decisions.

Model	*WER*	*UER*
5-gram FST	5.8	32.6
hier. RNN	**5.6**	**32.4**

Table 5. ASR LM fusion experiment results. The two systems in comparison differ only in the capitalization normalization model used to pre-process the LM training data.

5. CONCLUSIONS

Truecasing provides a factored solution to improve case-aware language modeling for applications such as ASR and text input in virtual keyboards. We propose a hierarchical word-and-character-based RNN model with the speed advantage of word-based models and accuracy advantage of character-based models. The model is efficient enough to be uploaded to mobile devices to train a language model using Federated Learning. The improvement is manifested in reduction of prediction error rates in a large-scale A/B experiment using a virtual keyboard and an ASR LM fusion experiment.

6. REFERENCES

[1] K. Nebhi, K. Bontcheva, and G. Gorrell, "Restoring capitalization in #tweets," in *Proc. WWW*, 2015.

[2] Y. Grishina, T. Gueudre, and R. Winkler, "Truecasing German user-generated conversational text," in *Proc. W-NUT*, 2020.

[3] L. V. Lita, A. Ittycheriah, S. Roukos, and N. Kambhatla, "tRuEcasIng," in *Proc. ACL*, 2003.

[4] A. Gravano, M. Jansche, and M. Bacchiani, "Restoring punctuation and capitalization in transcribed speech," in *Proc. ICASSP*, 2009.

[5] F. Beaufays and B. Strope, "Language model capitalization," in *Proc. ICASSP*, 2013.

[6] M. Sunkara, S. Ronanki, K. Dixit, S. Bodapati, and K. Kirchhoff, "Robust prediction of punctuation and truecasing for medical ASR," in *Proc. NLPMC*, 2020.

[7] M. Chen, A. T. Suresh, R. Mathews, A. Wong, C. Allauzen, F. Beaufays, and M. Riley, "Federated learning of n-gram language models," in *Proc. CoNLL*, 2019.

[8] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. AISTATS*, 2017.

[9] C. Chelba and A. Acero, "Adaptation of maximum entropy capitalizer: Little data can help a lot," in *Proc. EMNLP*, 2004.

[10] C. D. Santos and B. Zadrozny, "Learning character-level representations for pos tagging," in *Proc. ICML*, 2014.

[11] R. H. Susanto, H. L. Chieu, and W. Lu, "Learning to capitalize with character-level recurrent neural networks: An empirical study," in *Proc. EMMLP*, 2016.

[12] G. Ramena, D. Nagaraju, S. Moharana, D. Prasanna Mohanty, and N. Purre, "An efficient architecture for predicting the case of characters using sequence models," in *Proc. ICSC*, 2020.

[13] H. Zhang, R. Sproat, A. H. Ng, F. Stahlberg, X. Peng, K. Gorman, and B. Roark, "Neural models of text normalization for speech applications," *Comput. Linguistics*, vol. 45, no. 2, 2019.

[14] W. Wang, K. Knight, and D. Marcu, "Capitalizing machine translation," in *Proc. HLT/NAACL*, 2006.

[15] B. Nguyen, V. B. H. Nguyen, H. Nguyen, P. N. Phuong, T.-L. Nguyen, Q. T. Do, and L. C. Mai, "Fast and accurate capitalization and punctuation for automatic speech recognition using transformer and chunk merging," in *Proc. O-COCOSDA*, 2019.

[16] R. Rei, N. M. Guerreiro, and F. Batista, "Automatic truecasing of video subtitles using bert: A multilingual adaptable approach," in *Info. Proc. and Mgmt. of Uncertainty in Knowledge-Based Syst.*, 2020.

[17] C. Peyser, H. Zhang, T. N. Sainath, and Z. Wu, "Improving Performance of End-to-End ASR on Numeric Sequences," in *Proc. Interspeech*, 2019.

[18] B. Han, P. Cook, and T. Baldwin, "Lexical normalization for social media text," *ACM Trans. Intell. Syst. Technol.*, vol. 4, no. 1, 2013.

[19] P. Makarov and S. Clematide, "Semi-supervised contextual historical text normalization," in *Proc. ACL*, 2020.

[20] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. ICLR*, 2015.

[21] Y. Kim and A. M. Rush, "Sequence-level knowledge distillation," in *Proc. EMNLP*, 2016.

[22] J. Lichtarge, C. Alberti, S. Kumar, N. Shazeer, N. Parmar, and S. Tong, "Corpora generation for grammatical error correction," in *Proc. NAACL*, 2019.

[23] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, and P. Koehn, "One billion word benchmark for measuring progress in statistical language modeling," *CoRR*, vol. abs/1312.3005, 2013.

[24] R. Józefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, "Exploring the limits of language modeling," *CoRR*, vol. abs/1602.02410, 2016.

[25] T. Ouyang, D. Rybach, F. Beaufays, and M. Riley, "Mobile keyboard input decoding with finite-state transducers," *arXiv:1704.03987*, 2017.

[26] T. N. Sainath, Y. R. He, A. Narayanan, R. Botros, R. Pang, D. J. Rybach, C. Allauzen, E. Variani, J. Qin, Q.-N. Le-The, A. Gruenstein, A. Gulati, B. Li, C. Peyser, C.-C. Chiu, D. A. Caseiro, E. Guzman, I. C. McGraw, J. Yu, M. D. Riley, P. Rondon, Q. Liang, S. Mavandadi, S. yiin Chang, T. D. Strohman, W. R. Huang, W. Li, Y. Wu, and Y. Zhang, "An efficient streaming non-recurrent on-device end-to-end model with improvements to rare-word modeling," in *Proc. Interspeech*, 2021.

[27] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, "Conformer: Convolution-augmented transformer for speech recognition," in *Proc. Interspeech*, 2020.

[28] E. Variani, D. Rybach, C. Allauzen, and M. Riley, "Hybrid autoregressive transducer (hat)," in *Proc. ICASSP*, 2020.