Learning to Capitalize with Character-Level Recurrent Neural Networks: An Empirical Study

Raymond Hendy Susanto[†] and **Hai Leong Chieu**[‡] and **Wei Lu**[†]
[†]Singapore University of Technology and Design
[‡]DSO National Laboratories
{raymond_susanto,luwei}@sutd.edu.sg
chaileon@dso.org.sg

Abstract

In this paper, we investigate case restoration for text without case information. Previous such work operates at the word level. We propose an approach using character-level recurrent neural networks (RNN), which performs competitively compared to language modeling and conditional random fields (CRF) approaches. We further provide quantitative and qualitative analysis on how RNN helps improve truecasing.

1 Introduction

Natural language texts (e.g., automatic speech transcripts or social media data) often come in non-standard forms, and normalization would typically improve the performance of downstream natural language processing (NLP) applications. This paper investigates a particular sub-task in text normalization: case restoration or *truecasing*. Truecasing refers to the task of restoring case information (uppercase or lowercase) of characters in a text corpus. Case information is important for certain NLP tasks. For example, Chieu and Ng (2002) used unlabeled mixed case text to improve named entity recognition (NER) on uppercase text.

The task often presents ambiguity: consider the word "apple" in the sentences "he bought an apple" and "he works at apple". While the former refers to a fruit (hence, it should be in lowercase), the latter refers to a company name (hence, it should be capitalized). Moreover, we often need to recover the case information for words that are previously unseen by the system.

In this paper, we propose the use of character-level recurrent neural networks for truecasing. Previous approaches for truecasing are based on word level approaches which assign to each word one of the following labels: all lowercase, all uppercase, initial capital, and mixed case. For mixed case words, an additional effort has to be made to decipher exactly how the case is mixed (e.g., *MacKenzie*). In our approach, we propose a generative, character-based recurrent neural network (RNN) model, allowing us to predict exactly how cases are mixed in such words.

Our main contributions are: (i) we show that character-level approaches are viable compared to word-level approaches, (ii) we show that character-level RNN has a competitive performance compared to character-level CRF, and (iii) we provide our quantitative and qualitative analysis on how RNN helps improve truecasing.

2 Related Work

Word-based truecasing The most widely used approach works at the word level. The simplest approach converts each word to its most frequently seen form in the training data. One popular approach uses HMM-based tagging with an N-gram language model, such as in (Lita et al., 2003; Nebhi et al., 2015). Others used a discriminative tagger, such as MEMM (Chelba and Acero, 2006) or CRF (Wang et al., 2006). Another approach uses statistical machine translation to translate uncased text into a cased one. Interestingly, no previous work operated at the character level. Nebhi et al. (2015) investigated truecasing in tweets, where truecased cor-

pora are less available.

Recurrent neural networks Recent years have shown a resurgence of interest in RNN, particularly variants with long short-term memory (Hochreiter and Schmidhuber, 1997) or gated recurrent units (Cho et al., 2014). RNN has shown an impressive performance in various NLP tasks, such as machine translation (Cho et al., 2014; Luong et al., 2015), language modeling (Mikolov et al., 2010; Kim et al., 2016), and constituency parsing (Vinyals et al., 2015). Nonetheless, understanding the mechanism behind the successful applications of RNN is rarely studied. In this work, we take a closer look at our trained model to interpret its internal mechanism.

3 The Truecasing Systems

In this section, we describe the truecasing systems that we develop for our empirical study.

3.1 Word-Level Approach

A word-level approach truecases one word at a time. The first system is a tagger based on HMM (Stolcke, 2002) that translates an uncased sequence of words to a corresponding cased sequence. An N-gram language model trained on a cased corpus is used for scoring candidate sequences. For decoding, the Viterbi algorithm (Rabiner, 1989) computes the highest scoring sequence.

The second approach is a discriminative classifier based on linear chain CRF (Lafferty et al., 2001). In this approach, truecasing is treated as a sequence labeling task, labelling each word with one of the following labels: all lowercase, all uppercase, initial capital, and mixed case. For our experiments, we used the truecaser in Stanford's NLP pipeline (Manning et al., 2014). Their model includes a rich set of features (Finkel et al., 2005), such as surrounding words, character N-grams, word shape, etc.

Dealing with mixed case Both approaches require a separate treatment for mixed case words. In particular, we need a gazetteer that maps each word to its mixed case form – either manually created or statistically collected from training data. The character-level approach is motivated by this: Instead of treating them as a special case, we train our model to capitalize a word character by character.

3.2 Character-Level Approach

A character-level approach converts each character to either uppercase or lowercase. In this approach, mixed case forms are naturally taken care of, and moreover, such models would generalize better to unseen words. Our third system is a linear chain CRF that makes character-level predictions. Similar to the word-based CRF, it includes surrounding words and character N-grams as features.

Finally, we propose a character-level approach using an RNN language model. RNN is particularly useful for modeling sequential data. At each time step t, it takes an input vector x_t and previous hidden state h_{t-1}, and produces the next hidden state h_t. Different recurrence formulations lead to different RNN models, which we will describe below.

Long short-term memory (LSTM) is an architecture proposed by Hochreiter and Schmidhuber (1997). It augments an RNN with a memory cell vector c_t in order to address learning long range dependencies. The content of the memory cell is updated additively, mitigating the *vanishing gradient problem* in vanilla RNNs (Bengio et al., 1994). Read, write, and reset operations to the memory cell are controlled by input gate i, output gate o, and forget gate f. The hidden state is computed as:

$$
\begin{aligned}
i_t &= \sigma(W_i h_{t-1} + U_i x_t) && (1)\\
o_t &= \sigma(W_o h_{t-1} + U_o x_t) && (2)\\
f_t &= \sigma(W_f h_{t-1} + U_f x_t) && (3)\\
g_t &= \tanh(W_g h_{t-1} + U_g x_t) && (4)\\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t && (5)\\
h_t &= o_t \odot \tanh(c_t) && (6)
\end{aligned}
$$

where σ and tanh are element-wise sigmoid and hyperbolic tangent functions, and W_j and U_j are parameters of the LSTM for $j \in \{i, o, f, g\}$.

Gated recurrent unit (GRU) is a gating mechanism in RNN that was introduced by Cho et al. (2014). They proposed a hidden state computation with reset and update gates, resulting in a simpler LSTM variant:

$$
\begin{aligned}
r_t &= \sigma(W_r h_{t-1} + U_r x_t) && (7)\\
z_t &= \sigma(W_z h_{t-1} + U_z x_t) && (8)\\
\widetilde{h}_t &= \tanh(W_h(r_t \odot h_{t-1}) + U_h x_t) && (9)\\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \widetilde{h}_t && (10)
\end{aligned}
$$

	EN-Wikipedia				EN-WSJ				EN-Reuters				DE-ECI			
	Acc.	P	R	F_1	Acc.	P	R	F_1	Acc.	P	R	F_1	Acc.	P	R	F_1
Word-based Approach																
LM ($N = 3$)	94.94	89.34	84.61	86.91	95.59	91.56	78.79	84.70	94.57	93.49	79.43	85.89	95.67	97.84	87.74	92.51
LM ($N = 5$)	94.93	89.42	84.41	86.84	95.62	91.72	78.79	84.77	94.66	93.92	79.47	86.09	95.68	97.91	87.70	92.53
CRF-WORD	96.60	94.96	87.16	90.89	97.64	93.12	90.41	91.75	96.58	93.91	87.19	90.42	96.09	98.41	88.73	93.32
Chelba and Acero (2006)	n/a				97.10	-	-	-	n/a				n/a			
Character-based Approach																
CRF-CHAR	96.99	94.60	89.27	91.86	97.00	94.17	84.46	89.05	97.06	94.63	89.12	91.80	98.26	96.95	96.59	96.77
LSTM-SMALL	96.95	93.05	90.59	91.80	97.83	93.99	90.92	***92.43***	97.37	93.08	92.63	92.86	98.70	97.52	97.39	97.46
LSTM-LARGE	97.41	93.72	92.67	***93.19***	97.72	93.41	90.56	91.96	97.76	94.08	93.50	***93.79***	99.00	98.04	97.98	***98.01***
GRU-SMALL	96.46	92.10	89.10	90.58	97.36	92.28	88.60	90.40	97.01	92.85	90.84	91.83	98.51	97.15	96.96	97.06
GRU-LARGE	96.95	92.75	90.93	91.83	97.27	90.86	90.20	90.52	97.12	92.02	92.07	92.05	98.35	96.86	96.79	96.82

Table 2: Truecasing performance in terms of precision (P), recall (R), and F_1. All improvements of the best performing character-based systems (**bold**) over the best performing word-based systems (underlined) are statistically significant using sign test ($p < 0.01$). All improvements of the best performing RNN systems (*italicized*) over CRF-CHAR are statistically significant using sign test ($p < 0.01$).

At each time step, the conditional probability distribution over next characters is computed by linear projection of h_t followed by a softmax:

$$P(x_t = k|x_{1:t-1}) = \frac{\exp(w_k h_t)}{\sum_{j=1}^{|V|} \exp(w_j h_t)} \quad (11)$$

where w_k is the k-th row vector of a weight matrix W. The probability of a sequence of characters $x_{1:T}$ is defined as:

$$P(x_{1:T}) = \prod_{t=1}^{T} P(x_t|x_{1:t-1}) \quad (12)$$

Similar to the N-gram language modeling approach we described previously, we need to maximize Equation 12 in order to decode the most probable cased sequence. Instead of Viterbi decoding, we approximate this using a beam search.

4 Experiments and Results

4.1 Datasets and Tools

Our approach is evaluated on English and German datasets. For English, we use a Wikipedia corpus from (Coster and Kauchak, 2011), WSJ corpus (Paul and Baker, 1992), and the Reuters corpus from the CoNLL-2003 shared task on named entity recognition (Tjong Kim Sang and De Meulder, 2003). For German, we use the ECI Multilingual Text Corpus from the same shared task. Each corpus is tokenized.[1] The input test data is lowercased. Table 1 shows the statistics of each corpus split into training, development, and test sets.

We use SRILM (Stolcke, 2002) for N-gram language model training ($N \in \{3, 5\}$) and HMM decoding. The word-based CRF models are trained using the CRF implementation in Stanford's CoreNLP 3.6.0 (Finkel et al., 2005). We use a recommended configuration for training the truecaser.We use CRFSuite version 0.12 (Okazaki, 2007) to train the character-based CRF model. Our feature set includes character N-grams ($N \in \{1, 2, 3\}$) and word N-grams ($N \in \{1, 2\}$) surrounding the current character. We tune the ℓ_2 regularization parameter λ using a grid search where $\lambda \in \{0.01, 0.1, 1, 10\}$.

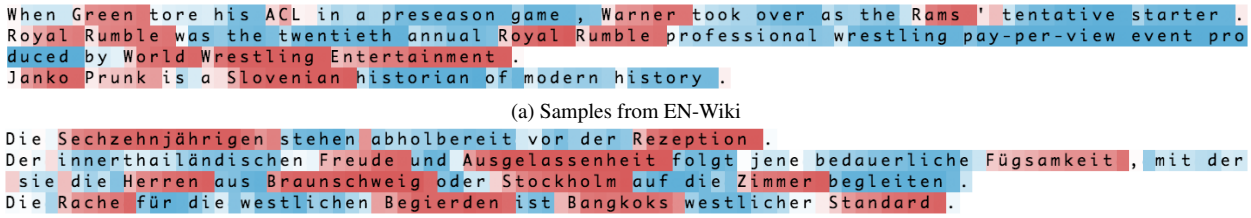
Corpus	Split	#words	#chars
EN-Wiki	train	2.9M	16.1M
	dev	294K	1.6M
	test	32K	176K
EN-WSJ	train	1.9M	10.5M
	dev	101K	555K
	test	9K	48K
EN-Reuters	train	3.1M	16.8M
	dev	49K	264K
	test	44K	231K
DE-ECI	train	2.8M	18M
	dev	51K	329K
	test	52K	327K

Table 1: Statistics of the data.

We use an open-source character RNN implementation.[2] We train a SMALL model with 2 layers and 300 hidden nodes, and a LARGE model with 3 layers and 700 hidden nodes. We also vary the hidden unit type (LSTM/GRU). The network is trained using truncated backpropagation for 50 time steps. We use a mini-batch stochastic gradient descent with batch size 100 and RMSprop update (Tieleman and Hinton, 2012). We use dropout regularization (Srivastava et al., 2014) with 0.25 probability. We choose the model with the smallest validation loss after 30 epochs. For decoding, we set beam size to 10. The experimental settings are reported in more depth in the supplementary materials. Our system and code are publicly available at `http://statnlp.org/research/ta/`.

[1] News headlines, which are all in uppercase, are discarded.

[2] `https://github.com/karpathy/char-rnn`



(a) Samples from EN-Wiki

(b) Samples from DE-ECI

Figure 1: Cells that are sensitive to lowercased and capitalized words. Text color represents activations ($-1 \leq \tanh(c_t) \leq 1$): positive is blue, negative is red. Darker color corresponds to greater magnitude.

4.2 Results

Table 2 shows the experiment results in terms of precision, recall, and F_1. Most previous work did not evaluate their approaches on the same dataset. We compare our work to (Chelba and Acero, 2006) using the same WSJ sections for training and evaluation on 2M word training data. Chelba and Acero only reported error rate, and all our RNN and CRF approaches outperform their results in terms of error rate.

First, the word-based CRF approach gives up to 8% relative F_1 increase over the LM approach. Other than WSJ, moving to character level further improves CRF by 1.1-3.7%, most notably on the German dataset. Long compound nouns are common in the German language, which generates many out-of-vocabulary words. Thus, we hypothesize that character-based approach improves generalization. Finally, the best F_1 score for each dataset is achieved by the RNN variants: 93.19% on EN-Wiki, 92.43% on EN-WSJ, 93.79% on EN-Reuters, and 98.01% on DE-ECI.

We highlight that different features are used in CRF-WORD and CRF-CHAR. CRF-CHAR only includes simple features, namely character and word N-grams and sentence boundary indicators. In contrast, CRF-WORD contains a richer feature set that is predefined in Stanford's truecaser. For instance, it includes word shape, in addition to neighboring words and character N-grams. It also includes more feature combinations, such as the concatenation of the word shape, current label, and previous label. Nonetheless, CRF-CHAR generally performs better than CRF-WORD. Potentially, CRF-CHAR can be improved further by using larger N-grams. The decision to use simple features is for optimizing the training speed. Consequently, we are able to dedicate more time for tuning the regularization weight.

Training a larger RNN model generally improves performance, but it is not always the case due to possible overfitting. LSTM seems to work better than GRU in this task. The GRU models have 25% less parameters. In terms of training time, it took 12 hours to train the largest RNN model on a single Titan X GPU. For comparison, the longest training time for a single CRF-CHAR model is 16 hours. Training LM and CRF-WORD is much faster: 30 seconds and 5.5 hours, respectively, so there is a speed-accuracy trade-off.

5 Analysis

5.1 Visualizing LSTM Cells

An interesting component of LSTM is its memory cells, which is supposed to store long range dependency information. Many of these memory cells are not human-interpretable, but after introspecting our trained model, we find a few memory cells that are sensitive to case information. In Figure 1, we plot the memory cell activations at each time step (i.e., $\tanh(c_t)$). We can see that these cells activate differently depending on the case information of a word (towards -1 for uppercase and +1 for lowercase).

5.2 Case Category and OOV Performance

Corpus	Lower	Cap.	Upper	Mixed	OOV
EN-Wiki	79.91	18.67	0.91	0.51	2.40
EN-WSJ	84.28	13.06	2.63	0.03	3.11
EN-Reuters	78.36	19.80	1.53	0.31	5.37
DE-ECI	68.62	29.15	1.02	1.21	4.01

Table 3: Percentage distribution of the case categories and OOV words

In this section, we analyze the system performance on each case category. First, we report the percentage distribution of the case categories in each test set in Table 3. For both languages, the most frequent case category is lowercase, followed by capitalization, which generally applies to the first word

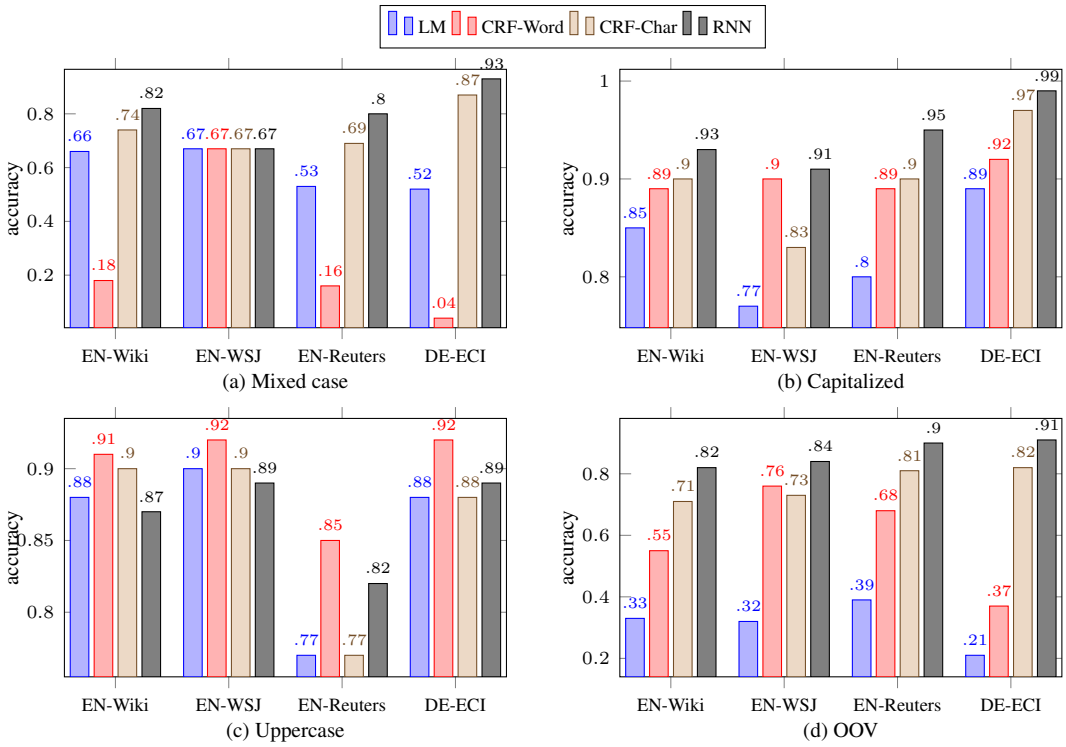


Figure 2: Accuracy on mixed case (a), capitalized (b), uppercase (c), and OOV words (d).

in the sentence and proper nouns. The uppercase form, which is often found in abbreviations, occurs more frequently than mixed case for English, but the other way around for German.

Figure 2 (a) shows system accuracy on mixed case words. We choose the best performing LM and RNN for each dataset. Character-based approaches have a better performance on mixed case words than word-based approaches, and RNN generally performs better than CRF. In CRF-WORD, surface forms are generated after label prediction. This is more rigid compared to LM, where the surface forms are considered during decoding.

In addition, we report system accuracy on capitalized words (first letter uppercase) and uppercase words in Figure 2 (b) and (c), respectively. RNN performs the best on capitalized words. On the other hand, CRF-WORD performs the best on uppercase. We believe this is related to the rare occurrences of uppercase words during training, as shown in Table 3. Although mixed case occurs more rarely in general, there are important clues, such as character prefix. CRF-CHAR and RNN have comparable performance on uppercase. For instance, there are only 2 uppercase words in WSJ that were predicted differently between CRF-CHAR and RNN. All systems perform equally well (∼99% accuracy) on lowercase. Overall, RNN has the best performance.

Last, we present results on out-of-vocabulary (OOV) words with respect to the training set. The statistics of OOV words is given in Table 3. The system performance across datasets is reported in Figure 2 (d). We observe that RNN consistently performs better than the other systems, which shows that it generalizes better to unseen words.

6 Conclusion

In this work, we conduct an empirical investigation of truecasing approaches. We have shown that character-level approaches work well for truecasing, and that RNN performs competitively compared to language modeling and CRF. Future work includes applications in informal texts, such as tweets and short messages (Muis and Lu, 2016).

Acknowledgments

We would also like to thank the anonymous reviewers for their helpful comments. This work is supported by MOE Tier 1 grant SUTDT12015008.

References

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Ciprian Chelba and Alex Acero. 2006. Adaptation of maximum entropy capitalizer: Little data can help a lot. *Computer Speech & Language*, 20(4):382–399.

Hai Leong Chieu and Hwee Tou Ng. 2002. Teaching a weaker classifier: Named entity recognition on upper case text. In *Proceedings of ACL*, pages 481–488.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of EMNLP*, pages 1724–1734.

William Coster and David Kauchak. 2011. Simple English Wikipedia: A new text simplification task. In *Proceedings of ACL-HLT*, pages 665–669.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of ACL*, pages 363–370.

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

Andrej Karpathy, Justin Johnson, and Fei-Fei Li. 2016. Visualizing and understanding recurrent networks. In *Proceedings of ICLR*.

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

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML*, pages 282–289.

Lucian Vlad Lita, Abe Ittycheriah, Salim Roukos, and Nanda Kambhatla. 2003. tRuEcasIng. In *Proceedings of ACL*, pages 152–159.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of EMNLP*, pages 1412–1421.

Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of ACL System Demonstrations*, pages 55–60.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proceedings of INTERSPEECH*, pages 1045–1048.

Aldrian Obaja Muis and Wei Lu. 2016. Weak semi-Markov CRFs for noun phrase chunking in informal text. In *Proceedings of NAACL*.

Kamel Nebhi, Kalina Bontcheva, and Genevieve Gorrell. 2015. Restoring capitalization in #tweets. In *Proceedings of WWW Companion*, pages 1111–1115.

Naoaki Okazaki. 2007. CRFsuite: A fast implementation of conditional random fields (CRFs).

Douglas B Paul and Janet M Baker. 1992. The design for the Wall Street Journal-based CSR corpus. In *Proceedings of the Workshop on Speech and Natural Language*, pages 357–362.

Lawrence R Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Andreas Stolcke. 2002. SRILM-an extensible language modeling toolkit. In *Proceedings of ICSLP*, pages 901–904.

Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2).

Erik F Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL*, pages 142–147.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Proceedings of NIPS*, pages 2755–2763.

Wei Wang, Kevin Knight, and Daniel Marcu. 2006. Capitalizing machine translation. In *Proceedings of NAACL-HLT*, pages 1–8.