International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426

Punctuation and Capitalization Restoration using Bi-LSTM Network

Ashish Bansal

Abstract: ***In this paper we present Bi-Directional Long short-term memory (BLSTM) network with linear Conditional Random Field (CRF) to restore punctuation and capitalization in an unsegmented speech transcript. Our approach will restore both punctuation and capitalization using a single model. This is purely lexical based approach with pre trained glove embeddings as an input. The task was treated as a sequence tagging problem where the input is sequence of un- punctuated and un-capitalized words, and the output is a corresponding sequence of punctuated and capitalized words. We demonstrated the accuracy of proposed model are competitive with the state-of-the-art models and can do both the tasks in a single model.***

Keywords: Punctuation prediction, Capitalization restoration, Neural network, true-casing, sentence segmentation

1. Introduction

Typically, the output transcript from naive Automatic Speech Recognition (ASR) system is un-punctuated, un-capitalized and unsegmented sequence of words. While these outputs are widely adopted in many applications relying on short utterances such as voice commands, voice assistants, dictation tools applications such as voice assistants, or voice-based commands. However, for downstream processing of traditional NLP classification tasks such as sentimental analysis or detecting language of certain documents (Emails, News or a website) which are trained on punctuated texts and absence of punctuation and capitalization can cause a substantive decrease in their performance. Even, it would increase human readability as well as improving the accuracy of post processing such as machine translation or language under- standing. Therefore, it is very important to restore Punctuation and Capitalization.

A lot of different approaches have been put to restore punctuation and then restoring capitalization automatically as independent tasks. Restoration of punctuation has been tackled by various approaches and methods such as sentence boundary detection, n-gram models, treating this task as a machine translation task, translating un-punctuated to punctuated text. In this paper we will focus on the approach of sequence tagging to restore punctuation and capitalization using single model.

In past research works, Substantially LSTM is proposed to model state of art Punctuation restoration system. Our re-search is to use BLSTM with CRF approach to restore both punctuation and capitalization in a single model. Our approach to build this model can be described in three steps:

- First, Bi-directional LSTM is proposed to account both past(left) and future(right) inputs and to model the relationship between input features and output labels.
- Secondly, we proposed Condition Random Field (CRF) layer on top of Bi-LSTM to achieve performance gain by catching contextual information for this task as expected in other sequence labelling tasks.
- Third, our research results showcase Bi-LSTM two-layer model with CRF can achieve state-of-the-art performance in punctuation and capitalization prediction.

Novelty of our approach is to use the single net- work (BLSTM + CRF) to restore punctuation and capitalization in a given unsegmented text. In above approach different data sets were transformed in the required format to solve the stated task. Our next section is a deep dive of our approach. Subsequent section goes in detail on models, datasets used, experimentation setup with metrics used as well as the results and finally concluding the paper with future work.

2. Method

Data Preparation

In this approach we dealt the task of inferring punctuation and capitalization in a given unsegmented text- specifically Commas, Period, Question mark, Quotes and Capitalization. This approach was generalized for multiple lines or a para-graph (raw transcript output from ASR). To attain this goal, we treated this problem as sequence tagging problem.

In our approach we have used different datasets such as CoNLL, IWSLT 2011, Europarl datasets to train our proposed model. All these datasets were read from the t text files. We defined 10 classes in total:

```
1. Japan japan <CAP>
2. began began O
3. the the O
4. defence defence O
5. of of O
6. their their O
7. Asian asian <CAP>
8. Cup cup <CAP>
9. title title O
10.with with O
11.a a O
12.lucky lucky O
13.2-1 2-1 O
14.win win O
15.against against O
16.Syria syria <CAP>
17.in in O
18.a a O
19.Group group <CAP>
20.championship championship O
21.match match O
22.on on O
23.Friday. friday <CAP_PERIOD>
```

Figure 1: An Example of a transformed sequence S for input

- O (means no Capitalization & Punctuation mark followed)
- CAP (means no punctuation mark followed by the word whose first letter in cap)
- COMMA (means comma (,) followed)
- PERIOD (means period (.) followed)
- QUESTIONMARK (means question mark(?) followed)
- QOUTATION (means quotation (“) followed)
- CAP PERIOD (means period (.) followed by the word whose first letter in cap)
- CAP COMMA (means comma (,) followed by the word
- CAP QUESTIONMARK (means question mark (?) followed by the word whose first letter in cap)
- CAP QOUTATION (means quotation (“) followed by the word whose first letter in cap)

While reading the text files for the sequences. In a particular sequence S, all tokens were transformed as shown in figure 1.

3. Model

Our model is a bidirectional LSTM based model with linear Conditional Random Field (CRF) where we treated punctuation and capitalization task as a sequence labelling task.

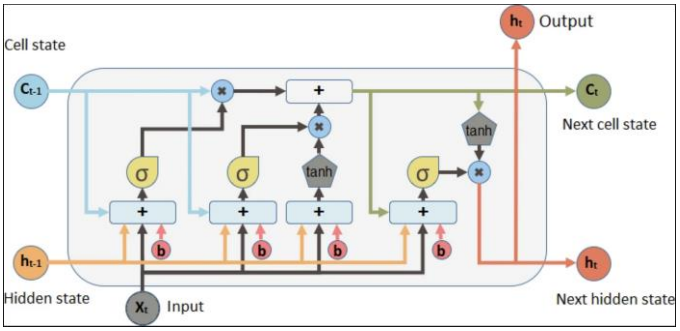

Figure 2: Figure

BLSTM

Recurrent neural network (RNN) that can handle the vanishing gradient problem is known as Long Short-Term Memory (LSTM) network. LSTM is also better at maintaining long-range connections and understanding the connection between values at the start and end of a sequence. By modifying a gating structure of a traditional RNN model, the LSTM model can learn or retain a longer data sequence. Therefore, LSTM has three gates: input, forget, and hidden. An LSTM unit with these three gates and a memory cell forms a layer of neural network neurons, and each neuron has a hidden layer and a current state. Figure 3 shows the LSTM cell’s structural layout.

The forget gate is used to specify whether or not certain data will be kept. This preservation is accomplished using the following formula;

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

where x_t denotes input at time t, h_{t-1} denotes the output of previous cell, and σ is a sigmoid function. If a forget gate outputs 1 (one), the information is stored in the cell state. The sigmoid function creates a vector in the following step. New possible values are stored in this vector. The updated values are specified by input gates, and the vector c_t is up- dated with possible new values. This new vector is evaluated with the following formulas;

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$
$$C'_t = tanh(W_c[h_{t-1}, x_t] + b_c)$$

Now cell’s old state c_t is updated to new cell state c_t

$$C_t = f_t * C_{t-1} + i_t * C'_t$$

Eventually, we select the network’s output regarding on the cell state. This selection process is carried out by using the following formulas;

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * tanh(C_t)$$

A special type of LSTM network which is used generally for natural language processing is called Bidirectional Long Short-Term Memory. In BiLSTM, there are two different LSTM networks in order to represent the input in both direction (backward and forward). Therefore, representing the input in both directions of the sequence, it is an effective tool for modeling the sequential relationships between words and phrases. In BiLSTM, two hidden states are introduced,

Paper ID: SR24809041216 DOI: https://dx.doi.org/10.21275/SR24809041216 2021

one for accessing the context of previous input and another one for accessing the context of the next input $\overleftarrow{h_t}$ and another one for accessing the context of the next input $\overrightarrow{h_t}$. Therefore, the formula for the hidden state of BiLSTM can be defined as:

$$h_t = \overrightarrow{h_t} \oplus \overleftarrow{h_t}$$

In summary, BiLSTM reverses the direction of information flow by adding one extra LSTM layer. It simply means that in the additional LSTM layer, the input sequence flows backward. The outputs from the two LSTM layers are then combined in a variety of ways, including average, sum, multiplication, and concatenation. BiLSTM process the input sequence of S = w_1, w_2,...,w_n sequentially. Here w_i denotes the i^{th} word of the sentence *S*. The embedding of each word w is supplied to BiLSTM model in order to produce the concatenated output X.

For many sequences labeling tasks, it is beneficial to have access to both past (left) and future (right) contexts. However, the LSTM's hidden state ht takes information only from past, knowing nothing about the future. An elegant solution whose effectiveness has been proven by previous work (Dyer et al., 2015) is bi-directional LSTM (BLSTM). The basic idea is to present each sequence forwards and backwards to two separate hidden states to capture past and future information, respectively. Then the two hidden states are concatenated to form the final output.

CRF Tagging Models

A very simple - but surprisingly effective—tagging model is to use the ht's as features to make independent tagging decisions for each output yt (Ling et al., 2015b). Despite this model's success in simple problems like POS tagging, its independent classification decisions are limiting when there are strong dependencies across output labels. NER is one such task, since the "grammar" that characterizes interpretable sequences of tags imposes several hard constraints (e.g., I-PER cannot follow B-LOC; see §2.4 for details) that would be impossible to model with independence assumptions. Therefore, instead of modeling tagging decisions independently, we model them jointly using a conditional random field (Lafferty et al., 2001).

Conditional random field (CRF) is a statistical model well suited for handling NER problems, because it takes context into account. In other words, when a CRF model makes a prediction, it factors in the impact of neighbouring samples by modelling the prediction as a graphical model. For example, a linear chain CRF is a popular type of a CRF model, which assumes that the tag for the present word is dependent only on the tag of just one previous word (this is somewhat similar to Hidden Markov Models, although CRF's topology is an undirected graph).

One problem with the linear chain CRFs (Figure 1) is that they are capable of capturing the dependencies between labels in the forward direction only. If the model encounters an entity like "Johns Hopkins University" it will likely tag the Hopkins token as a name, because the model is "blind" to the university token that appears downstream. One way to resolve this challenge is to introduce a bidirectional LSTM (BiLSTM) network between the inputs (words) and the CRF.

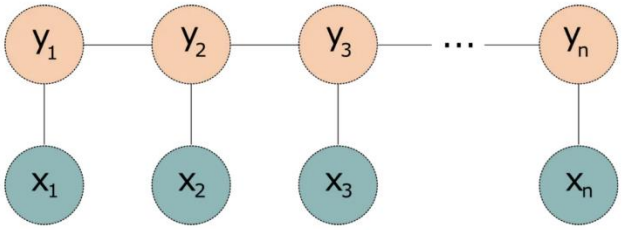

Figure 3: A simple linear- chain conditional random fields model. The model takes an input sequence x (words) and target sequence y (IOB tags)

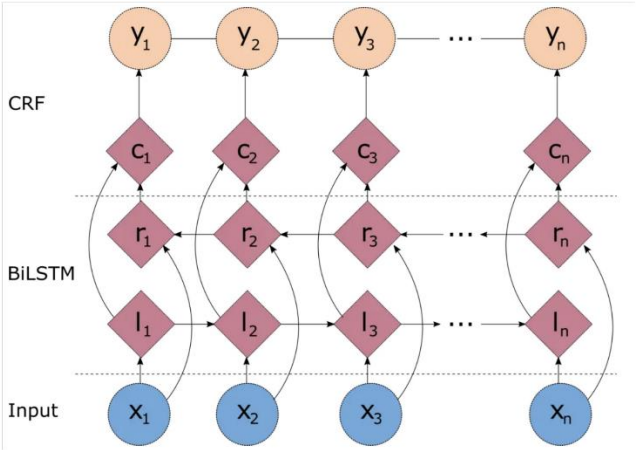

Figure 4: Architecture of a BiLSTM- CRF model

BLSTM-CRF

The bidirectional LSTM consists of two LSTM networks - one takes the input in a forward direction, and a second one taking the input in a backward direction. Combining the outputs of the two networks yields a context that provides information on samples surrounding each individual token. The output of the BiLSTM is then fed to a linear chain CRF, which can generate predictions using this improved context. This combination of CRF and BiLSTM is often referred to as a BiLSTM-CRF model (Lample et al 2016), and its architecture is shown in Figure 2.

Finally, we construct our neural network model by feeding the output vectors of BLSTM into a CRF layer. At first each word, its represented in the word embeddings using bert and then with the word embedding vector to feed into the BLSTM network. Finally, the output vectors of BLSTM are fed to the CRF layer to jointly decode the best label sequence. Dropout layers are applied on both the input and output vectors of BLSTM. Experimental results show that using dropout significantly improve the performance of our model.

Firstly, every word in sentence x is represented as a vector which includes the word's character embedding and word embedding. The character embedding is initialized randomly. The word embedding usually is from a pre-trained word embedding file. All the embeddings will be fine-tuned during the training process. Second, the inputs of BiLSTM-CRF model are those embeddings and the outputs are predicted labels for words in sentence x. Although, it is not necessary

Paper ID: SR24809041216 DOI: https://dx.doi.org/10.21275/SR24809041216

to know the details of BiL STM layer, in order to understand the CRF layer more easily, we have to know what is the meaning of the output of BiLSTM Layer. The picture above illustrates that the outputs of BiLSTM layer are the scores of each label. For example, for w0the outputs of BiLSTM node are 1.5 (B-Person), 0.9 (I-Person), 0.1 (B-Organization), 0.08 (I-Organization) and 0. These scores will be the inputs of the CRF layer.

Then, all the scores predicted by the BiLSTM blocks are fed into the CRF layer. In the CRF layer, the label sequence which has the highest prediction score would be selected as the best answer.

Model Implementation

We constructed our neural network by feeding the output vectors of BLSTM to linear chain CRF to decode the best output tag sequence. Architecture of our complete network presented in Figure3. To construct this there are three main steps involved as follows:

- **Word Embeddings:** First step was to use a dense word representation for each word $\boldsymbol{w} \in R^n$ to capture the meaning and relevant features for our task. Glove pre trained embeddings were used to represent the word. For each word a vector was built by concatenation of glove word embeddings $w_{glove} \in R^{d1}$ of dimension 300 and the vector containing features extracted from the character level. BLSTM was run over the sequence of character embeddings $\boldsymbol{w}_{chars} \in R^{d2}$. Each character c_i of a word $\boldsymbol{w}$ =[c_1,...,c_p] is associated to a vector $c_i \in R^{d3}$. BLSTM was run over the sequence of character embed dings and concatenate the final states to obtain a fixed-size vector $\boldsymbol{w}_{chars} \in R^{d2}$, this vector captures the morphol ogy of the word. Word char embeddings $\boldsymbol{w}_{chars}$ were the concatenated to the glove word embedding $\boldsymbol{w}_{glove}$ to get a vector representing our word. Implementation of the network to compute char embeddings is represented in Figure2.
- Once we have our word representation, we simply run a BLSTM over the sequence of word vectors and obtain an other sequence of vectors and concatenating of the two hidden states in the case of a BLSTM, $h \in R^k$. We use the hidden states of each time step and not just the final states. Thus, we had as input a sequence of m word vec tors $\boldsymbol{w}_1$,...,$\boldsymbol{w}_m \in R^n$ and now we have a sequence of vectors h_1,...,$h_m \in R_k$ Whereas the $\boldsymbol{w}_t$ only captured information at the word level (syntax and semantics), the ht also take context into account.
- Final step is to decode, at this stage, each word $\boldsymbol{w}$ is as sociated to a vector $\boldsymbol{h}$ that captures information from the meaning of the word, its characters and its context. We have 10 classes. We take a matrix $W \in R^{10x\,k}$ and compute a vector of scores $s \in R^10 = W. h+ b$. We can interpret the s [i] (i-th component of s) as the score of class i for word $\boldsymbol{w}$. To decode the final scores we use Lin ear chain CRF. Given a sequence of words w_1,...,w_m, a sequence of score vectors s_1,...,s_m and a sequence of tags y_1,...,y_m, a linear-chain CRF defines a global score C∈ R such that

$$C(y_1, \ldots, y_m) = begin + scores + transitions + end \quad (1)$$

$$C(y_1, \ldots, y_m) = b[y_1] + \sum_{t=1}^{m} s_t[y_t] S + \sum_{t=1}^{m-1} T[y_t, y_{t+1}] + e[y_m] \quad (2)$$

where T is a transition matrix in R^{10x10} and e, b ∈ R^{10} are vectors of scores that capture the cost of beginning or ending with a given tag. The use of the matrix T captures linear (one step) dependencies between tagging decisions. Example of scoring a sequence using a linear chain-CRF shown in Figure 4.

Two steps have to be followed to decode a sentence sequence:

- First is to compute the scores of all the 10^m tagging choices to choose the best one or normalize each sequence score into a probability. Let's suppose of a solution $\tilde{s}_{t+1}(y^{t+1})$ for time steps $t + 1$,...,m for sequence that start with yt+1 for each of the 10 possibilities. The solution

$$\tilde{s}_t(y_t) = argmax_{y_t,\ldots,y_m} C(y_t, \ldots, y_m) \quad (3)$$

$$\tilde{s}_t(y_t) = argmax_{y_{t+1}} s_t[y_t] + T[y_t, y_{t+1}] + \tilde{s}_{t+1}(y^{t+1}) \quad (4)$$

- The final step in decoding using Linear chain CRF is to apply the softmax to the scores of all possible sequences to get the probability P(y) of all the sequence of y tags. Partition factor (Sum of all the possible sequence scores) is computed to achieve the probability as

$$Z = \sum_{y_1,\ldots,y_m} e^{C(y_1,\ldots,y_m)} \quad (5)$$

We will sum over all the possible paths. Let's Zt(yt) be the sum of all the sequence at time step t with tag yt.

$$Z_t(y_t) = \sum_{y_{t+1}} e^{s_t[y_t] + T[y_t, y_{t+1}]} \sum_{y_{t+2},\ldots,y_m} e^{C(y_{t+1},\ldots,y_m)} \quad (6)$$

$$Z_t(y_t) = \sum_{y_{t+1}} e^{s_t[y_t] + T[y_t, y_{t+1}]} Z_{t+1}(y_{t+1}) \quad (7)$$

$$\log Z_t(y_t) = \log \sum_{y_{t+1}} e^{s_t[y_t] + T[y_t, y_{t+1}] + \log Z_{t+1}(y_{t+1})} \quad (8)$$

The probability of a given sequence of tags would look like:

$$\mathbb{P}(y_1, \ldots, y_m) = \frac{e^{C(y_1,\ldots,y_m)}}{Z} \quad (9)$$

4. Experiment

Datasets

To restore punctuation and capitalization we chose three different datasets to train and test our model. We used Europarl v7, IWSLT2011(TED Talks) and CoNLL datasets. These all three datasets are in English language and widely used in the similar tasks by different researchers. All these datasets concatenated and are tagged by the tagging scheme discussed in above section.

Metrics

To assess the impact on performance of the amount of training data available, we prepared three data sets with varying numbers of tokens, and evaluated the trained models on above

mentioned datasets. We employ Precision, Recall, and F1 score as key metrics to assess the effectiveness of our proposed model. Precision gauges the model's capability to accurately identify relevant
entities, while Recall evaluates its ability to capture all relevant entities within a dataset. The F-score, on the other hand, serves as the harmonic mean of Precision and Recall, offering a balanced measure of the model's overall performance.

Detailed calculations for these metrics are provided below.

$$Precision = \frac{TP}{TP + FP} \quad (10)$$

$$Recall = \frac{TP}{TP + FN} \quad (11)$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2*TP + FP + FN} \quad (12)$$

where True Positive (TP) represents entities correctly recognized by the model and that align with annotated entities. False Positive (FP) pertains to entities erroneously identified by the model, which do not align with annotated entities. Conversely, False Negative (FN) encompasses annotated entities that the model fails to recognize.

5. Results

Results on the custom classes we have. All values are in percentages.

Tag	Precision	Recall	F1 Store
CAP	94	92	93
CAP COMMA	74	65	69
CAP PERIOD	91	82	87
CAP QUOTATION	86	77	81
CAP QUESTION MARK	79	54	64
COMMA	79	66	72
PERIOD	97	95	96
QUOTATION	95	95	95
QUESTION MARK	91	83	87
Other	97	99	98

6. Conclusions and Future Work

We have presented an approach to punctuation and capitalization restoration using text-based models Bi-directional LSTM network with linear Conditional Random field (CRF) models. Our results suggest that using the approach of IOB tagging with larger training datasets leads to consistent improvements in performance and a simple framework to restore Punctuation and Capitalization. Furthermore, we show that low-frequency symbols such as question marks and dashes are much harder to model using simple n-grams than commas and periods. A natural direction to continue this research is to use more advanced models in literature such as transformers and expanding the punctuation classes to present more under-represented punctuations in the unsegmented text, and also include other textual features (e.g. part-of-speech and shallow syntactic information), as well as acoustic/prosodic features from the audio signal (e.g. pause duration and word final intonation).

References

[1] E. Shriberg, A. Stolcke, D. Hakkani-T ur, and G. T ur, "Prosodybased automatic segmentation of speech into sentences and topics," Speech Comm., vol. 32(1-2), pp. 127–154, 2000.
[2] Y. Liu, E. Shriberg, A. Stolcke, D. Hillard, M. Ostendorf, and M. Harper, "Enriching speech recognition with automatic detection of sentence boundaries and disfluencies," IEEE Trans. Audio Speech Lang. Process., vol. 14(5), pp. 1526–1540, 2006.
[3] Y. Gotoh and S. Renals, "Sentence boundary detection in broadcast speech transcripts," in ISCA Workshop on Automatic Speech Recognition, 2000.
[4] D. Beeferman, A. Berger, and J. Lafferty, "Cyberpunc: A lightweight punctuation annotation system for speech," in ICASSP, 1998, vol. 2, pp. 689–692.
[5] E.W. Brown and A.R. Coden, "Capitalization recovery for text," IR Techniques for Speech Applications, 2002.
[6] H. Christensen, Y. Gotoh, and S. Renals, "Punctuation an- notation using statistical prosody models," in ISCA Work- shop on Prosody in Speech Recognition and Understanding, 2001.
[7] J. Huang and G. Zweig, "Maximum entropy model for punctuation annotation from speech," in ICSLP, 2002.
[8] B. Favre, R. Grishman, D. Hillard, H. Ji, D. Hakkani-Tur, and M. Ostendorf, "Punctuating Speech for Information Extraction," in ICASSP, 2008.
[9] C. Chelba and A. Acero, "Adaptation of maximum entropy capitalizer: Little data can help a lot," Computer Speech and Language, vol. 20(4), pp. 382–399, 2006.
[10] Wei Wang, Kevin Knight, and Daniel Marcu, "Capitaliz- ing machine translation," in HLT/ACL, 2006.
[11] T. Brants, A.C. Popat, P. Xu, F.J. Och, and J. Dean, "Large language models in Machine Translation," in EMNLP, 2007.
[12] Akakpo Agbago, Roland Kuhn, George Foster. 2005. Truecasing for the Portage system. In Recent Advances in Natural Language Processing (RANLP), Borovets, Bul- garia.
[13] Douglas E. Appelt, Jerry R. Hobbs, John Bear, David Is- rael, Megumi Kameyama, Andy Kehler, David Martin, Karen Myers, Mabry Tyson. 1995. SRI International FAS- TUS system MUC-6 Test Results and Analysis. In Pro- ceedings of the 6th Message Understanding Conference.
[14] Łukasz Augustyniak, Piotr Szymanski, Mikołaj Morzy, Piotr Zelasko, Adrian Szymczak, Jan Mizgajski, Yishay Carmiel, Najim Dehak. Punctuation Prediction in Spontaneous Conversations: Can We Mitigate ASR Errors with Retrofitted Word Embeddings? arXiv:2004.05985 [cs.CL].
[15] Alexei Baevski, Sergey Edunov, Yinhan Liu, Luke Zettle- moyer, Michael Auli. 2019. Cloze-driven Pretraining of Self-attention Networks. arXiv:1903.07785.
[16] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. arXiv:1409.0473.20
[17] Timothy Baldwin, Paul Cook, Marco Lui, Andrew MacKinlay, Li Wang. 2013. How noisy social media

text,how diffrnt social media sources? In Proceedings of the 6th International Joint Conference on Natural Language Processing, 356-364.

[18] Fernando Batista, Isabel Trancoso, Nuno Mamede. 2009. Automatic Recovery of Punctuation Marks and Capital- ization Information for Iberian Languages. In Proceed- ings of the Joint SIG-IL/Microsoft Workshop on Speech and Language Technologies for Iberian Languages, Porto Salvo, Portugal, 99-102.

[19] Europarl: A Parallel Corpus for Statistical Machine Translation, Philipp Koehn, MT Summit 2005, pdf.