Toward Multi-domain Language Generation using Recurrent Neural Networks

Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Lina M. Rojas-Barahona,
Pei-Hao Su, David Vandyke, Steve Young
Cambridge University Engineering Department,
Trumpington Street, Cambridge, CB2 1PZ, UK
{thw28,mg436,nm480,lmr46,phs26,djv27,sjy}@cam.ac.uk

Abstract

In this paper we study the performance and domain scalability of two different Neural Network architectures for Natural Language Generation in Spoken Dialogue Systems. We found that by imposing a sigmoid gate on the dialogue act vector, the Semantically Conditioned Long Short-term Memory generator can prevent semantic repetitions and achieve better performance across all domains compared to an RNN Encoder-Decoder generator. However, in a domain adaptation experiment, the RNN Encoder-Decoder generator, with a separate slot and value parameterisation, is capable of learning faster by leveraging out-of-domain data. We conclude that the way to represent and integrate the semantic elements is of great importance to NN-based NLG systems. Further advances will therefore require a representation that is more scalable across domains without significantly compromising in-domain performance.

本文研究了语音对话系统中自然语言生成的两种神经网络体系结构的性能和领域可扩展性

我们发现，通过在对话行为向量上施加一个sigmoid门，语义条件长短时记忆生成器可以防止语义重复，并在所有域上取得比RNN编码器-解码器生成器更好的性能。

然而，在一个域适应实验中，RNN编码器-解码器生成器，具有单独的插槽和值参数化，能够通过利用域外数据学习更快

研究结果表明，语义元素的表示和集成方法对基于神经网络的神经语言系统具有重要意义。因此，进一步的改进将需要一种跨领域更可伸缩的表示，而不显著影响域内性能。

1 Introduction

Over the past decades, significant progress has been made in applying statistical methods to automate the development of Spoken Dialogue Systems (SDS) [1, 2]. However, relatively less effort has been put into the Natural Language Generation (NLG) component because of the difficulty of data collection. As a consequence, rule-based [3, 4], or a hybrid of rule-based plus statistical approaches [5, 6] remain the norm for most systems.

Recently, the rise of deep learning and the use of crowdsourcing platforms such as Amazon Mechanical Turk offer a great opportunity for data-driven NLG. Word-based generation was first proposed by Oh and Rudnicky [7], in which a set of language models (LM) were trained for each utterance class and used to generate utterances in an over-generation and reranking paradigm. Phrase-based generators such as Bagel [8] have been observed to improve performance over class-based LM. However, training such generators requires the alignment between utterances and semantics to have been annotated beforehand, which is nontrivial for untrained workers. More recently, Recurrent Neural Networks (RNN) have been proposed for learning generation decisions end-to-end. Wen et al [9] used an RNN generator accompanied by a set of NN-based rerankers to generate utterances. Their input features are gated by simple heuristics in order to prevent undesirable repetitions in the generated output. Subsequently, a Semantically Conditioned Long Short-term Memory generator (SC-LSTM) [10] was proposed which learnt the gating signal and LM jointly. The authors showed the deep extension of this model achieved state-of-the-art performance for the two datasets used.

Another generation idea, borrowed from Machine Translation (MT), is the RNN Encoder-Decoder architecture with attention mechanism [11]. Since the model is so general that the encoder and decoder can be chosen separately given the task at hand, many people have adopted it to solve a

variety of problems, e.g. image captioning [12] and object recognition [13]. Mei et al [14] proposed a similar Encoder-Decoder architecture by using two layers of attention to model content selection and attentive input aggregation. They evaluated the model on selective generation datasets such as WEATHERGOV and ROBOCUP and obtained a large performance boost compared to previous methods. One goal of this paper is to study the pros and cons of these neural network-based generators by comparing them with each other on the same dataset.

Modern design of dialogue systems usually starts with a well-defined domain ontology, such as restaurant search or hotel booking. However, the goal of building an open domain dialogue system that is capable of talking about any topic is still far away. Several works have begun to address this problem, such as [15] for multi-domain dialogue state tracking, or [16] for multi-domain dialogue management. In this paper we also move toward this goal by analysing the domain scalability of neural language generators. We trained general models by pooling all datasets altogether and assessed the model's ability to learn from multi-domain datasets. We also tested whether generators can be extended to a new, unseen domain by using only a limited amount of in-domain data.

2 The Neural Language Generator

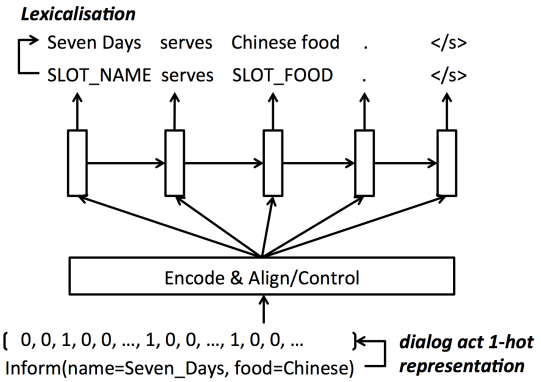


Figure 1: An unrolled view of the RNN-based neural language generator. The output part is an RNN decoder while the encoder is subject to various designs.

In general, a neural language generator can be divided into two components, an encoder to incorporate the target meaning representation as the model input, and a decoder to generate sentences, as shown in Figure 1. Although the decoder is typically an RNN, the choice of the encoder is made case by case because it depends not only on the nature of the meaning representation but also on the interaction between semantic elements. Once the input meaning representation is encoded, an attention-based or feature controlling mechanism is used to select and aggregate the input semantic elements. At generation time, the state of the RNN decoder is conditioned by the aggregated input vector, and the output distribution of the RNN is sampled to obtain the next token[1]. Finally, a lexicalisation operation is performed and the final realisation is obtained.

2.1 Attention-based RNN Encoder-Decoder

The RNN Encoder-Decoder architecture was first proposed in the MT literature [11]. The encoder first encodes the source information into a distributed vector representation, which the decoder subsequently decodes into the target output. By adopting the idea from Mei et al [14] and modifying it based on our task, on the encoder side we used a separate parameterisation of slots and values. We embed each slot-value pair into a distributed vector representation $\mathbf{z}_i$ by,

$$\mathbf{z}_i = \mathbf{s}_i + \mathbf{v}_i \tag{1}$$

[1]We use *token* instead of *word* because our model operates on text for which slot names and values have been delexicalised.

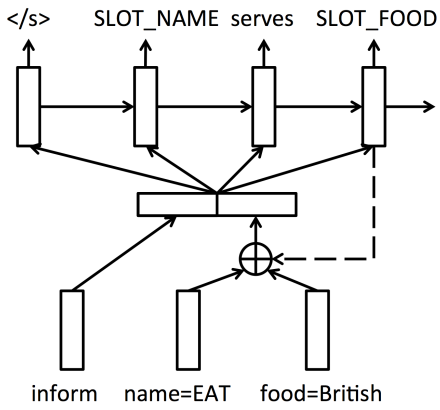


Figure 2: The Encoder-Decoder used for NLG in this paper. The input side is a dialogue act (DA) embedding with an attention over slot-value pairs while the output side is a LSTM network. Note the action type embedding is not included in the attention mechanism since it acts more like a background signal to control the style of the sentence. Attention is recomputed for each time step t.

where $\mathbf{s}_i$ and $\mathbf{v}_i$ are the i-th slot and value embedding, respectively. i runs over the given slot-value pairs. Finally the dialogue act (DA) embedding at each time step t is formed by,

$$\mathbf{d}_t = \mathbf{a} \oplus \sum_i \omega_{t,i} \mathbf{z}_i \tag{2}$$

where $\mathbf{a}$ is the act type embedding, $\oplus$ is vector concatenation, and $\omega_{t,i}$ is the weight of i-th slot-value pair calculated by the attention mechanism,

$$\beta_{t,i} = \mathbf{q}^\intercal \tanh(\mathbf{W}_{hm}\mathbf{h}_{t-1} + \mathbf{W}_{mm}\mathbf{z}_i) \tag{3}$$

$$\omega_{t,i} = e^{\beta_{t,i}} / \sum_i e^{\beta_{t,i}} \tag{4}$$

where $\mathbf{h}_{t-1}$ is the last step hidden layer. $\mathbf{q}$, $\mathbf{W}_{hm}$, and $\mathbf{W}_{mm}$ are parameters to learn. The DA embedding $\mathbf{d}_t$ is then fed into a standard LSTM as additional information to produce the hidden layer,

$$\begin{pmatrix} \mathbf{i}_t \\ \mathbf{f}_t \\ \mathbf{o}_t \\ \hat{\mathbf{c}}_t \end{pmatrix} = \begin{pmatrix} \text{sigmoid} \\ \text{sigmoid} \\ \text{sigmoid} \\ \tanh \end{pmatrix} \mathbf{W}_{4n,3n} \begin{pmatrix} \mathbf{w}_t \\ \mathbf{h}_{t-1} \\ \mathbf{d}_t \end{pmatrix}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

where n is the hidden layer size, $\mathbf{w}_t$ is the input word embedding, and $\mathbf{W}_{4n,3n}$ is the model parameter. The objective for training the model is maximising the log-likelihood of the training corpus.

2.2 Semantically Conditioned LSTM

The Gated RNN generator was first proposed by Wen et al [9]. Their model used a 1-hot vector for each slot-value pair and applied a heuristic controlling gate for each pair in the vector, by exactly matching the slot-value pair and its corresponding token in the realisation. The Semantically Conditioned LSTM [10] embraces the same idea, but instead of using the heuristic gates, the authors proposed a model based on an LSTM architecture that can learn both the LM and the gating control signal jointly. The central idea is to apply a sigmoid gate at each time step to the dialogue act which decides whether to keep or drop slot-value pairs. The gate value depends on the generation history. The additional gate is therefore dubbed the reading gate [10] since it acts like a keyword and key-phrase detector that can learn to associate certain patterns of generated tokens with certain

slot-value pairs. The entire architecture is end-to-end differentiable so that back propagation can be used to optimise the parameters. The objective function contains not only the log-likelihood of the training corpus but also the regularisation terms on the DA transition dynamics [10].

Although the SC-LSTM has been shown to achieve state-of-the-art performance on in-domain datasets, problems remain as to whether the model can generalise to unseen domains. The current slot-value pair based parameterisation forms the major obstacle to this. Since the slot is bundled with the value using the same parameter [10], the model can not generalise on the same values that are associated with different slots, and vice versa. E.g. *price_range=don't_care* is considered to be totally different from *food=don't_care*, even though they may have a very similar realisation.

3 Experiments

3.1 Experimental Setup

We conducted our experiments on three dialogue domains: restaurant, hotel, and laptop. The restaurant and hotel domains were collected in Wen et al [10]. These two domains are relatively small and multiple references were collected for each dialogue act. In order to test the ability of the systems on a more complicated domain with a richer semantic input space, we crowdsourced another domain, laptop, by collecting only one sentence for each dialogue act via Amazon Mechanical Turk (AMT) service. Each worker was presented a dialogue act (DA), represented by an act type with a set of slot-value pairs, and asked to enter an appropriate natural language realisation. This yielded about 13K utterances, one for each DA, which is much more difficult than the previous two domains (5.1K utterances, $\sim$200 distinct DAs). The new laptop ontology is shown in Table 1.

The generators were implemented using the Theano library [17, 18], and trained by partitioning each of the collected corpus into a training, validation, and testing set in the ratio 3:1:1. All the generators were trained by treating each sentence as a mini-batch. Apart from the standard log-likelihood objective, an l_2 regularisation term was added to the objective function for every 10 training examples. The hidden layer size was set to be 80 for all cases, and deep networks, structured as in Wen et al [10], were trained with two hidden layers and a 50% dropout rate. Stochastic gradient descent and back propagation through time [19] were used to optimise the parameters. In order to prevent overfitting, early stopping was implemented using the validation set.

During decoding, we over-generated 20 utterances and selected the top 5 realisations for each DA according to the following reranking criteria,

$$R = -(F(\theta) + \lambda \mathrm{ERR}) \tag{5}$$

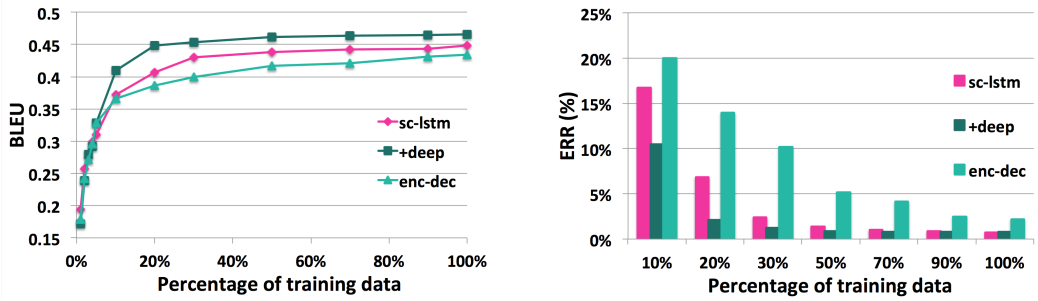
where λ is a tradeoff constant, $F(\theta)$ is the cost generated by network parameters θ, and the slot error rate ERR is computed by exact matching of the slot tokens in the candidate utterances. λ is set to a large value (10) in order to severely penalise nonsensical outputs. Since our generator works stochastically and the trained networks can differ depending on the initialisation, all the results shown below were averaged over 5 randomly initialised networks. The BLEU-4 metric [20] and slot error rate (ERR) [10], which is calculated by averaging slot errors over each of the top 5 realisations in the entire corpus, were used for the corpus-based evaluation. We used multiple references to compute BLEU score whenever they were available (i.e. for restaurant & hotel). We compared three models, SC-LSTM (*sc-lstm*), its deep version (+*deep*), and Encoder-Decoder generator (*enc-dec*).

Table 1: Laptop ontology

act type	inform, inform_only_match, inform_no_match, inform_count, inform_all, inform_no_info, recommend, compare, confirm, select, suggest, request, request_more, goodbye
slots	family*, battery_rating*, drive_range*, **is_for_business***, price_range*, weight_range*, warranty, battery, design, dimension, utility, weight, platform, memory, price, drive, processor

bold=binary slots, *=slots can take don't care value

3.2 Model Comparison



(a) Comparison of BLEU

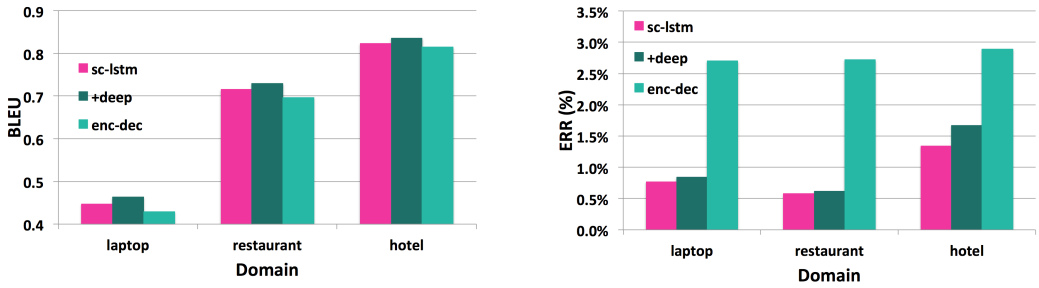
(b) Comparison of slot error rate

Figure 3: Performance comparison of the three approaches when training from scratch.

In our first experiments we trained each of the three models (*sc-lstm*,+*deep*, and *enc-dec*) from scratch on the laptop domain with a varied proportion of training data, starting from 10% to 100%. The result is shown in Figure 3. It clearly shows BLEU increases and the slot error rate decreases as the models are trained on more data. We found the *sc-lstm* and its deep version +*deep* are clearly better than the Encoder-Decoder *enc-dec* in all cases. The performance of the best model +*deep* starts to saturate around 50% while the *enc-dec* seems to continue improving even when all the training set is used. Furthermore, the *enc-dec* has a much greater slot error rate comparing to the SC-LSTM approaches (*sc-lstm* & +*deep*).

3.3 General Models

We then trained general models by pooling all the data from the three domains together and tested them in each individual domain. The results are shown in Figure 4. We found *sc-lstm* and +*deep* consistently outperformed *enc-dec* on all three domains. Although the BLEU score is comparable, we observed that *enc-dec* has difficulties in driving down the slot error rate in all cases.



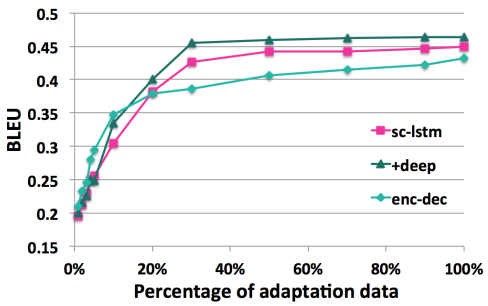
(a) BLEU on three domains

(b) Slot error rate on three domains

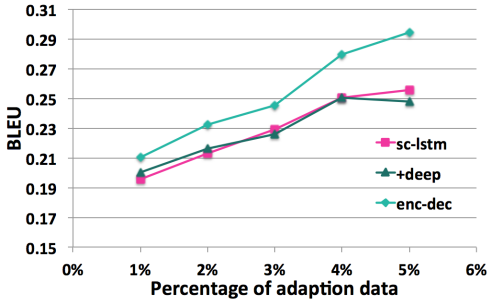
Figure 4: Comparison of general models on different domains.

3.4 Adaptation

A preliminary experiment on domain scalability was also conducted. To test whether the three models can leverage out-of-domain data, we first trained out-of-domain models by pooling the restaurant and hotel domain datasets together. After that, we varied the proportion of in-domain training data (laptop dataset) and fine tuned the model parameters. The results can be seen in Figure 5. Not surprisingly, the SC-LSTM approaches (*sc-lstm* & +*deep*) still outperform the *enc-dec* model when a sufficient amount of in-domain data is used (>20%), as suggested in Figure 5 (a). However, when only a limited amount of in-domain data is available (<5%), the *enc-dec* model tends to adapt to the new domain faster than the SC-LSTMs. This phenomenon is highlighted in Figure 5 (b).



(a) BLEU with varied amount of adaptation data



(b) BLEU with 1%-5% of adaptation data

Figure 5: Performance on laptop domain when adapting models trained on restaurant+hotel dataset.

3.5 Discussion

According to these preliminary experiments, we can draw the following conclusions:

1. Since the SC-LSTMs (*sc-lstm* & *+deep*) can effectively prevent undesirable semantic repetitions by gating the DA vector, it effectively drives down the slot error rate. On the other hand, the Encoder-Decoder generator (*enc-dec*) controls this by putting a distribution over all possible slot-value embeddings, which may still cause undesirable repetitions. This causes the SC-LSTMs to consistently have lower slot error rate than the Encoder-Decoder.

2. The bundled parameterisation of slot-value pairs of SC-LSTMs is necessary to learn a good alignment between slot-value pairs and their realisations. Unfortunately, it also limits the scalability of the model since the weights of different slots with the same value are not shared, and vice versa. This tradeoff can be seen in the preliminary adaptation result shown in Figure 5. With a separate parameterisation of slots and values, the Encoder-Decoder generator can adapt faster in the beginning. However, as the amount of in-domain data increases, the SC-LSTMs outperform the Encoder-Decoder since their bundled slot-value pair parameterisation allows them to learn better alignments.

3. Although the Encoder-Decoder generator shows signs of better generalisation, none of the models here benefited significantly from the out-of-domain data. More work should be done to balance the strengths and weakness of the two models.

4 Conclusion

In this paper we compared two RNN-based generators, the Semantically Conditioned LSTM generator and the RNN Encoder-Decoder generator. We found that in general, the SC-LSTM approaches outperformed the RNN Encoder-Decoder because the SC-LSTMs can better prevent the undesirable semantic repetitions. However, in the preliminary adaptation experiment the RNN Encoder-Decoder shows a better capability in extending to another domain when only a very limited amount of in-domain data is available. The separate parameterisation of slots and values is the key to this. Future work will study better ways of representing and integrating slots and values so that the model can leverage out-of-domain sources without significantly sacrificing its in-domain performance.

Acknowledgments

Tsung-Hsien Wen and David Vandyke are supported by Toshiba Research Europe Ltd, Cambridge Research Laboratory.

References

[1] Steve Young, Milica Gašić, Blaise Thomson, and Jason D. Williams. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 2013.

[2] Matthew Henderson, Blaise Thomson, and Steve Young. *Proceedings of SIGdial*, chapter Word-Based Dialog State Tracking with Recurrent Neural Networks. Association for Computational Linguistics, 2014.

[3] Adam Cheyer and Didier Guzzoni. Method and apparatus for building an intelligent automated assistant, 2007. US Patent App. 11/518,292.

[4] Danilo Mirkovic and Lawrence Cavedon. Dialogue management using scripts, 2011. EP Patent 1,891,625.

[5] Irene Langkilde and Kevin Knight. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the 36th Annual Meeting of the ACL*, ACL '98, 1998.

[6] Marilyn A Walker, Owen C Rambow, and Monica Rogati. Training a sentence planner for spoken dialogue using boosting. *Computer Speech and Language*, 2002.

[7] Alice H. Oh and Alexander I. Rudnicky. Stochastic language generation for spoken dialogue systems. In *Proceedings of the 2000 ANLP/NAACL Workshop on Conversational Systems - Volume 3*, ANLP/NAACL-ConvSyst '00, 2000.

[8] François Mairesse and Steve Young. Stochastic language generation in dialogue using factored language models. *Computer Linguistics*, 2014.

[9] Tsung-Hsien Wen, Milica Gašić, Dongho Kim, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking. In *Proceedings of SIGdial*. Association for Computational Linguistics, 2015.

[10] Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Proceedings of EMNLP*. Association for Computational Linguistics, 2015.

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[12] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015.

[13] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. *CoRR*, abs/1406.6247, 2014.

[14] Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. What to talk about and how? selective generation using lstms with coarse-to-fine alignment. *CoRR*, abs/1509.00838, 2015.

[15] Nikola Mrksic, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gasic, Pei-hao Su, David Vandyke, Tsung-Hsien Wen, and Steve J. Young. Multi-domain dialog state tracking using recurrent neural networks. *CoRR*, abs/1506.07190, 2015.

[16] Milica Gašić, Dongho Kim, Pirros Tsiakoulis, and Steve Young. Distributed dialogue policies for multi-domain statistical dialogue management.

[17] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference*, 2010.

[18] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[19] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 1990.

[20] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on ACL*. Association for Computational Linguistics, 2002.