Data Augmentation Using Many-To-Many RNNs for Session-Aware Recommender Systems

Martín Baigorria Alonso*
martinbaigorria@gmail.com

Figure 1: Multi-Destinations Trip Recommender on Booking.com

ABSTRACT

The ACM WSDM WebTour 2021 Challenge organized by Booking.com focuses on applying Session-Aware recommender systems in the travel domain. Given a sequence of travel bookings in a user trip, we look to recommend the user's next destination. To handle the large dimensionality of the output's space, we propose a many-to-many RNN model, predicting the next destination chosen by the user at every sequence step as opposed to only the final one. We show how this is a computationally efficient alternative to doing data augmentation in a many-to-one RNN, where we consider every subsequence of a session starting from the first element. Our solution achieved 4th place in the final leaderboard, with an accuracy@4 of 0.5566.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → *Neural networks.*

KEYWORDS

recommender systems; recurrent neural networks; session-aware recommendations

1 INTRODUCTION

Session-Aware recommender systems model the sequential decision process of a user in the context of a session, also considering past user actions or attributes [16, 23]. These systems have been essential for the growth of many e-commerce and content companies that need to organize a vast catalog of options into a relevant and easily manageable subset by the user [2, 4, 10, 11, 18].

The ACM WSDM WebTour 2021 Challenge organized by Booking.com [3] proposes a Session-Aware recommender systems problem that focuses on using a dataset of booking sequences and contextual information to make the best possible recommendation for a user in real-time. The quality of the recommendations is measured using recall@4. Booking.com currently has a recommender system for this problem in production (e.g., Figure 1), which enforces once again the importance of the problem.

Hidasi et al. [5] has recently proposed the use of Recurrent Neural Network (RNN) based models to overcome some of the difficulties other factor models or neighborhood-based approaches can have when modeling sparse sequential data. These models have been successful in other domains such as speech recognition [6] and natural language processing [12, 21, 24].

This manuscript proposes an approach to handling a high dimensional output space in RNN based Session-Aware recommender systems. Our main contributions to this problem include extending the many-to-one configuration typically-used RNN-based recommender systems to the many-to-many configuration. Instead of predicting the next booking by a user given a sequence of bookings, we predict the next booking at every time step. We show how this is a computationally efficient alternative to doing data augmentation in a many-to-one RNN, where we consider every subsequence of a session starting from the first element. Our empirical results show how this model can outperform the many-to-one RNNs by a significant margin. We also show how this extension biases the learning problem towards shorter trips and propose a correction by weighting the model's loss function.

We organize the rest of the paper as follows: In Section 2, we first give an overview of the challenge, the dataset provided by Booking.com, and the evaluation metric to be optimized. We then describe the many-to-many RNN architecture we used. We focus on explaining our design choices' motivations and the approach we followed for training and inference (in Section 3). Lastly, in Section 4 we discuss both the implementation details we used to iterate quickly and our experiments' results. Our code and documentation are available at https://github.com/mbaigorria/booking-challenge-2021-recsys.

2 CHALLENGE OVERVIEW

The ACM WSDM WebTour 2021 challenge proposed by Booking.com is an instance of the multi-destinations trip planning problem. Using an anonymized dataset of bookings, the goal is to recommend the next possible destination to a user in the context of a session.

2.1 Data

The dataset is composed of over a million anonymized reservations. Each hotel reservation is a part of a user's trip (identified by utrip_id) and includes multiple features such as the city, country, check-in, and checkout times. In the test set, these trips include at least four consecutive reservations. An overview of the dataset statistics is shown in Table 1.

*Paper submitted as an independent researcher.

Type	Users	Sessions	Cities	Accomodations	Session length		
					Min	Max	Median
Train	200,153	217,686	39,901	1,166,835	1	48	5
Test	68,502	70,662	21,305	378,667	4	44	5

Table 1: Descriptive statistics of the training and test sets

For each session in the test set, the city and country of the last booking are concealed. However, at inference time, other variables such as the check-in day of the next booking are also available.

2.2 Evaluation

For every trip in the test set, the correct city to be recommended is concealed. The metric used to evaluate the recommendations was precision@4. We can understand this metric as the percentage of times the correct city was in the top 4 recommendations for each trip. For this problem, the metric is equivalent to the recall@4, since there is only one relevant item to recommend per trip.

3 APPROACH & MOTIVATION

We designed our approach to be sequence and distance aware. Predicting the next destinations is a problem that must capture the notion of physical distance between cities and the sequential nature of the user's decision-making process. We should also be able to handle variable-length user trips. It is also worth remembering user trips may be censored, as users do not necessarily book every trip on the platform.

Given some highly dimensional categorical variables such as affiliate_id or city_id, we should also learn latent representations (also known as embeddings) to explicitly avoid using one-hot encodings. The large dimensionality of the output space can also be challenging. In this specific case, there are 39,901 possible cities to recommend. In the training set, only 52% of these cities appear as the last element of a trip. If we are not careful, any softmax-based model to predict the sequence's final booking is unlikely to make all cities predictable.

In Section 3.1, we explain how we tackled these challenges, the problems and trade-offs we faced, and some interesting future research directions.

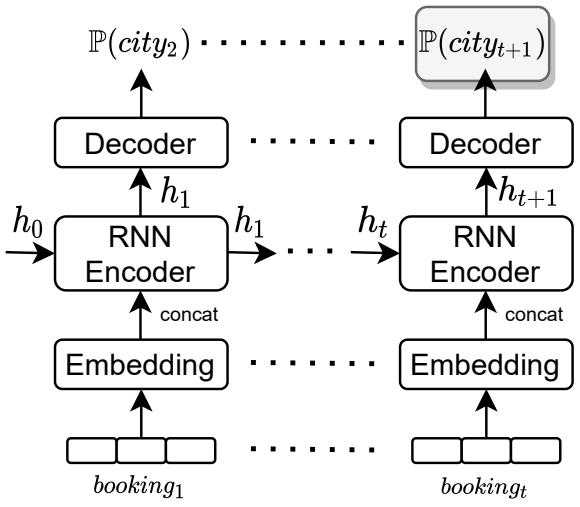

Figure 2: Many-to-many RNN based model architecture

3.1 Architecture

Our neural network architecture can be divided into three components. First, our model concatenates the embeddings of all features for each user trip booking. These concatenated vectors are then fed into a many-to-many RNN encoder, outputting the next city's probability mass function given the previous cities at every time step. We finally recommend the top 4 cities in the probability mass function of the final step. You can see our general architecture diagram in Figure 2.

3.1.1 *Feature engineering and representation.*

Feature engineering. Using the provided dataset, we created 14 features by using the available information of both the current and the next bookings. We were cautious to not introduce any type of data leakage. The variables used were:

- City and country of the current booking.
- Country in which the current booking made.
- Country in which the next booking is made.
- Check-in day, month, and year of the current booking.
- Check-in day of the next booking.
- Duration of stay (days) of the current booking.
- Duration of stay (days) of the next booking.
- Device class of the current booking.
- Transition days between a checkout and the next check-in.
- Current and next booking affiliate_id (e.g. direct, third-party referral, paid search engine, etc.).

The transition days between bookings can potentially help us capture censored bookings. The model can learn that an extended transition time could mean the user booked on another platform, depending on the city to be recommended.

Feature representation. We embed all categorical and numerical features. Numerical features have a low cardinality, so they should be easily learned by the model. This allowed us not to have to worry about feature scaling. The embedding dimension of all numerical features is set to 10 and for categorical features to 25 except for the device_id (5) and the city_id (128). To input these features into our model, we then concatenate all the embeddings at each time step. We also tried to merge all feature embeddings through multiplication, but our model degraded. This is similar to the findings by Mizrahi et al. for a similar dataset [14].

3.1.2 *Encoder.*

We use a 2 layered RNN encoder to represent the context of the user at every sequence time step. In a nutshell, an RNN can process a variable length sequence as follows:

$$h_t = g(h_{t-1}, f_t, \theta) \tag{1}$$

where g is a recurrent transition function parameterized by θ (e.g. GRU, LSTM), h_t is the hidden state at time t and f_t is the input vector at time t (the concatenated booking embeddings in our case). From the hidden state at time t, we can not only get the hidden state at $t+1$, but can also use a decoder to generate a probability mass function over cities in t+1. We experiment with both GRU and LSTM encoders [1, 7]. As others have previously pointed out, we found that GRUs tend to outperform LSTMs in some recommendation settings [5, 14].

Instead of using a many-to-one RNN, we decided to use a many-to-many architecture to predict the next booked city at each time step. This many-to-many architecture has the same number of parameters as the many-to-one architecture. We apply a decoder over all the hidden states for each sequence instead of just the last one. This is identical to training a many-to-one architecture with an augmented dataset containing all session subsequences beginning from the first element in a single batch. However, for a session of length n, the many-to-one architecture must do n forward passes with a number of sequential operations in $O(n^2)$ to get all the hidden states. On the other hand, the many-to-many model computes the same hidden states but linearly in n.

We apply recurrent dropout with a probability of 0.1 to both of the recurrent encoders. We also use dropout on the concatenated input embeddings with a probability of 0.3. This is done not only to regularize our model [19] but also to model the censorship in the data by partially omitting some features in the different time steps.

3.1.3 ***Decoder.*** The decoder maps the hidden state h_t of the RNN encoder every time step t to a probability mass function over the 39,901 cities. We have experimented with the following two parameterizations of the decoder:

(1) **Feedforward Neural Network**: This layer is parameterized with an input size equal to the number of hidden units in the RNN encoder and an output size equal to the number of cities. We then apply a softmax to the final output at each time step to get a probability mass function over cities.
(2) **Tied weights between city embeddings and output layer**: We set the dimension of the encoder hidden state h_t to be equal to the dimension of the city embeddings. We can then understand the encoder as learning a vector in a similar manifold as the cities. We then calculate the product between the hidden states and the transposed embedding matrix of cities, applying a softmax after to get a probability mass function over all cities at every time step. This is equivalent to parameterizing the feedforward encoder mentioned above with the weights of the city embedding matrix. Some implementations of Word2Vec [13] use a similar trick, where the input word and context word embeddings are parameterized with the same weights. It has recently been shown that this can also be interpreted as a form of data augmentation [8].

We concluded that tying the city embeddings with the output layer does not only lead to a faster training time because the network has a lower number of parameters, but it also has a better performance across different encoder configurations. You can see Table 3 for more details.

3.2 Loss function

We calculate the cross-entropy loss of the probability mass function over cities at every sequence step for each trip. Since we train the model in batches, we then average all these cross-entropy losses for every trip in a batch.

We have validated that the longer a user trip is, the easier it is for our model to predict the next booking. Table 2 shows the sequence length distribution for different datasets once we remove the last observation in every sequence. We do this since we do not have information about the next booking. The last two datasets concatenate both the training and the test set. To clarify, concatenating both the training and the test sets provided in the challenge is possible since the actual leaderboard targets are concealed.

Session length	Train	Test	Train + Test	Train + Test subsequences
1	0.001	-	0.001	0.243
2	0.003	-	0.113	0.243
3	0.452	0.453	0.398	0.215
4	0.229	0.231	0.204	0.119
5	0.126	0.127	0.114	0.069
6	0.074	0.076	0.066	0.041
7	0.044	0.042	0.040	0.025
8	0.027	0.027	0.024	0.016
9	0.016	0.016	0.015	0.010
10	0.010	0.010	0.009	0.006
> 10	0.028	0.027	0.026	0.006
Total sequences	217,573	70,662	288,235	**1,186,491 (>4x more)**

Table 2: Session length distribution for different types of datasets.

When considering all subsequences, the session length distribution becomes more concentrated on shorter sessions. This type of data augmentation comes at a cost since the underlying data generating process may be different for short and long trips.

Shorter sequences will also dominate the gradient estimates for each batch. We can correct this bias by weighting our loss function's cross-entropy outputs by the reverse cumulative frequency of the dataset sequence length. This way, each length's sequences would have the same contribution to gradient updates as in the non-augmented dataset. Our experiments in Table 3 show weighting the loss function degraded the model's performance. However, this could be related to how we distributed the trips in a batch to train the model. This will be further discussed in Section 4.2.

It is also worth pointing out that augmenting the dataset explicitly has the risk of data leakage, using a validation trip that our model has already seen during training time. The many-to-many model avoids this by design, processing all subsequences of a session in one forward pass inside a single batch.

3.3 Training and inference

We trained a stratified ten-fold average cross-validation ensemble using the Adam optimizer [9] with a learning rate of 10^{-3} and a batch size of 256. We trained each model for 50 epochs and used the recall@4 on the validation fold to pick the best model.

We obtained some additional data by unifying the training and the test set. We were careful to split the dataset into folds with a similar sequence length distribution. We considered the last probability mass function at inference time and extracted the top 4 recommendations with the maximum probability, precisely as in a many-to-one model.

4 EXPERIMENTAL RESULTS & DISCUSSION

In this section, we dive into the implementation details and empirical evaluations of our proposed recommendation approach.

4.1 Implementation

Our model was implemented in PyTorch [15], using an NVIDIA® Tesla® V100 GPU for training. The following approaches allowed us to lower the training time per model to below 10 minutes. We trained with a large batch size of 256, sorting the sequences by length before batching to avoid zero padding. We did not explore the hyperparameters we used thoroughly, from the embedding sizes to the learning rate and batch size. We believe additional performance gains are possible by optimizing these hyperparameters.

We pre-loaded all batches in GPU, and when doing 10-fold cross-validation, we were careful to keep the folds as balanced as possible, also shuffling the batches at every epoch. Using a many-to-many RNN architecture instead of doing explicit data augmentation allowed us to process four times the amount of sequences at roughly the same computational cost.

Model Type	Recurrent Unit	Tie Encoder & Decoder	Weight Type	Accuracy@4
Many To Many	GRU	True	UNWEIGHTED	**0.5345***
			WEIGHTED	0.5313*
		False	UNWEIGHTED	0.5202
			WEIGHTED	0.5196
		True	WEIGHTED	0.5119
			UNWEIGHTED	0.5094
	LSTM	True	UNWEIGHTED	0.5166
			WEIGHTED	0.5125
		False	UNWEIGHTED	0.5161
			WEIGHTED	0.5128
Many To One	GRU	True	WEIGHTED	0.5064
			UNWEIGHTED	0.5048
		False	WEIGHTED	0.5005
			UNWEIGHTED	0.4996
	LSTM	True	WEIGHTED	0.5078
			UNWEIGHTED	0.5026
		False	UNWEIGHTED	0.5071
			WEIGHTED	0.5017

Table 3: Results for different configurations of our model. Models with an asterisk correspond to an average ten-fold cross-validation ensemble.

4.2 Results

Based on the implementation discussed in Section 4, we measured the performance of our model's different configurations by building a new test set of 5,600 trips with 24,400 bookings from the training set. This test set had a similar trip length distribution as the challenge organizers' original test set.

Our results confirm GRUs seem to perform better than LSTMs for this task. We can also see the benefit of using many-to-many RNNs, which outperform many-to-one RNNs across all configurations. Tying the encoder and decoder weights also led to higher performance.

It is natural to wonder why the weighted version of the model did not perform as well as the unweighted version. We believe that this is related to our implementation. When we sorted the trips to avoid zero-padding, we biased each gradient update's mean and variance. Even if every batch has a fixed number of trips, this does not imply that it has the same number of augmented sequences. This can be solved using variable-sized batches, or avoiding sorting altogether.

5 CONCLUSIONS

One of the main challenges in Session-Aware recommender systems is effectively handling a large dimensional output space. This paper described how a many-to-many RNN could outperform the many-to-one configuration in such a setting. We showed how this extension is equivalent to doing data augmentation in a many-to-one RNN, with the pitfall of potentially biasing our gradient estimates. We finally showed the advantages of tying the feature embedding weights with the parameterization of the decoder.

The effect of having an uneven sequence length distribution in every batch and how to effectively handle the bias introduced by considering all subsequences in the many-to-many architecture can be an interesting future research direction. Attempting to do multi-task learning at every time step by also predicting other future attributes like the country might further boost our predictive performance.

In the dataset provided by Booking.com, around 12% of the users had multiple bookings. We attempted to model this by concatenating user trips with a separator token without success. This context can potentially be modeled using Hierarchical RNNs, or Transformer based architectures [17, 20, 22].

ACKNOWLEDGMENTS

We want to thank Sole Pera, Roberto Martín Pozzer, Leonardo Baldassini, Fabricio Previgliano, and Charlie Giudice for their useful comments on an earlier version of this work. We would also like to thank the ACM WSDM WebTour 2021 workshop organizers for the opportunity to participate in such an exciting challenge.

REFERENCES

[1] Kyunghyun Cho, Bart van Merriënboer, 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 the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1724–1734. https://doi.org/10.3115/v1/D14-1179

[2] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems* (Boston, Massachusetts, USA) *(RecSys '16)*. Association for Computing Machinery, New York, NY, USA, 191–198. https://doi.org/10.1145/2959100.2959190

[3] Dmitri Goldenberg, Kostia Kofman, Pavel Levin, Sarai Mizrachi, Maayan Kafry, and Guy Nadav. 2021. Booking.com WSDM WebTour 2021 Challenge. https://www.bookingchallenge.com/. In *ACM WSDM Workshop on Web Tourism (WSDM WebTour'21)*.

[4] Carlos A. Gomez-Uribe and Neil Hunt. 2016. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.* 6, 4, Article 13 (Dec. 2016), 19 pages. https://doi.org/10.1145/2843948

[5] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1511.06939

[6] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97. https://doi.org/10.1109/MSP.2012.2205597

[7] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[8] Hakan Inan, Khashayar Khosravi, and Richard Socher. 2017. Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=r1aPbsFle

[9] Diederik P Kingma and Jimmy Lei Ba. 2015. Adam: A method for stochastic gradient descent. In *ICLR: International Conference on Learning Representations*. 1–15.

[10] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* 7, 1 (2003), 76–80. https://doi.org/10.1109/MIC.2003.1167344

[11] James McInerney, Benjamin Lacker, Samantha Hansen, Karl Higley, Hugues Bouchard, Alois Gruson, and Rishabh Mehrotra. 2018. Explore, Exploit, and Explain: Personalizing Explainable Recommendations with Bandits. In *Proceedings of the 12th ACM Conference on Recommender Systems* (Vancouver, British Columbia, Canada) *(RecSys '18)*. Association for Computing Machinery, New York, NY, USA, 31–39. https://doi.org/10.1145/3240323.3240354

[12] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

[13] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger (Eds.). 3111–3119. https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html

[14] Sarai Mizrachi and Pavel Levin. 2019. Combining Context Features in Sequence-Aware Recommender Systems. In *RecSys (Late-Breaking Results)*. 11–15.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[16] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. 2018. Sequence-Aware Recommender Systems. *CoRR* abs/1802.08452 (2018). arXiv:1802.08452 http://arxiv.org/abs/1802.08452

[17] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. 2017. Personalizing Session-Based Recommendations with Hierarchical Recurrent Neural Networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems* (Como, Italy) *(RecSys '17)*. Association for Computing Machinery, New York, NY, USA, 130–137. https://doi.org/10.1145/3109859.3109896

[18] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 1994. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. ACM Press, 175–186.

[19] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15, 1 (2014), 1929–1958.

[20] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 1441–1450.

[21] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215* (2014).

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[23] Shoujin Wang, Longbing Cao, and Yan Wang. 2019. A Survey on Session-based Recommender Systems. *CoRR* abs/1902.04864 (2019). arXiv:1902.04864 http://arxiv.org/abs/1902.04864

[24] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR* abs/1609.08144 (2016). arXiv:1609.08144 http://arxiv.org/abs/1609.08144