Why not be Versatile? Applications of the SGNMT Decoder for Machine Translation

Felix Stahlberg[†] fs439@cam.ac.uk
Danielle Saunders[†] ds636@cam.ac.uk
Gonzalo Iglesias[‡] giglesias@sdl.com
Bill Byrne[‡†] bill.byrne@eng.cam.ac.uk, bbyrne@sdl.com
[†]Department of Engineering, University of Cambridge, UK
[‡]SDL Research, Cambridge, UK

Abstract

SGNMT is a decoding platform for machine translation which allows paring various modern neural models of translation with different kinds of constraints and symbolic models. In this paper, we describe three use cases in which SGNMT is currently playing an active role: (1) *teaching* as SGNMT is being used for course work and student theses in the MPhil in Machine Learning, Speech and Language Technology at the University of Cambridge, (2) *research* as most of the research work of the Cambridge MT group is based on SGNMT, and (3) *technology transfer* as we show how SGNMT is helping to transfer research findings from the laboratory to the industry, eg. into a product of SDL plc.

1 Introduction

The rate of innovation in machine translation (MT) has gathered impressive momentum over the recent years. The discovery and maturation of the neural machine translation (NMT) paradigm (Sutskever et al., 2014; Bahdanau et al., 2015) has led to steady and substantial improvements of translation performance (Williams et al., 2014; Jean et al., 2015; Luong et al., 2015; Chung et al., 2016; Wu et al., 2016; Gehring et al., 2017; Vaswani et al., 2017). Fig. 1 shows that this progress is often driven by significant changes in the network architecture. This volatility poses major challenges in MT-related research, teaching, and industry. Researchers potentially spend a lot of time implementing to keep their setups up-to-date with the latest models, teaching needs to identify suitable material in a changing environment, and the industry faces demanding speed requirements on its deployment processes. Another practical challenge many researchers are struggling with is the large number of available NMT tools (van Merriënboer et al., 2015; Junczys-Dowmunt et al., 2016; Klein et al., 2017; Sennrich et al., 2017; Helcl and Libovický, 2017; Bertoldi et al., 2017; Hieber et al., 2017).[1] Committing to one particular NMT tool bears the risk of being outdated soon, as keeping up with the pace of research is especially costly for NMT software developers.

The open-source SGNMT (Syntactically Guided Neural Machine Translation) decoder[2] (Stahlberg et al., 2017b) is our attempt to mediate the effects of the rapid progress in

[1]See `https://github.com/jonsafari/nmt-list` for a complete list of NMT software.
[2]Full documentation available at `http://ucam-smt.github.io/sgnmt/html/`.

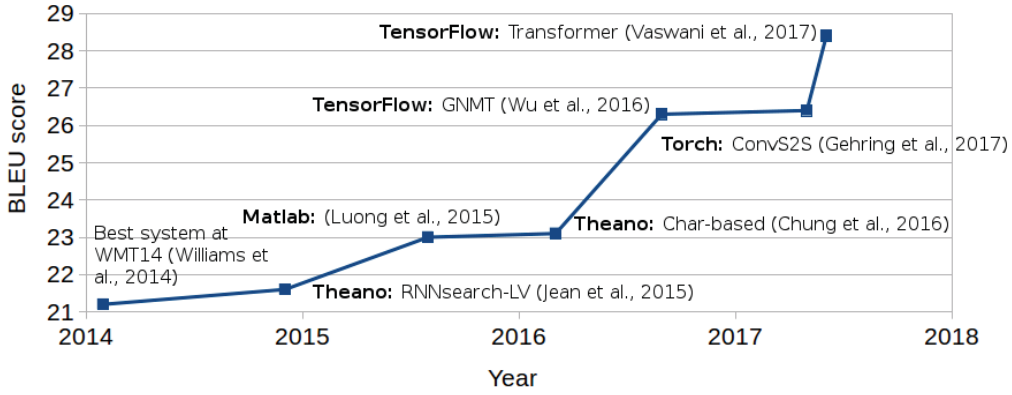

Figure 1: Best systems on the English-German WMT *news-test2014* test set over the years (BLEU script: Moses' `multi-bleu.pl`).

MT and the diversity of available NMT software. SGNMT introduces the concept of *predictors* as abstract scoring modules with left-to-right semantics. We can think of a *predictor* as an interface to a particular neural model or NMT tool. However, the interface also allows to implement constraints like in lattice or n-best list rescoring, and symbolic models such as n-gram language models or counting models as predictors. Our software architecture is designed to facilitate the implementation of new predictors. Therefore, SGNMT can be extended to a new model or tool with very limited coding effort because rather than reimplementing models it is often enough to access APIs within an adapter predictor.[3] Software packages which are not written in Python can be exposed in SGNMT if they have a Python interface.[4] Once a new predictor is implemented, it can be directly combined with all other predictors which are already available in SGNMT. Therefore, general techniques like lattice and n-best list rescoring (Stahlberg et al., 2016; Neubig et al., 2015), ensembling, MBR-based NMT (Stahlberg et al., 2017a), etc. only need to be implemented once (as predictor), and are automatically available for all models. This does not only speed up the transition to a new NMT toolkit, it also allows the combination of different NMT implementations, eg. ensembling a Theano-based NMT model (van Merriënboer et al., 2015) with a TensorFlow-based Tensor2Tensor (Google, 2017) model. Hasler et al. (2017) demonstrated the versatility of SGNMT by combining five very different models (RNN LM, feedforward NPLM, Kneser-Ney LM, bag-to-seq model, seq-to-seq model) and a bag-of-words constraint using predictors.

Not only the way scores are assigned to translations is open for extension in SGNMT (via predictors), but also the search strategy (*decoder*) itself. Decoders in SGNMT are defined upon the predictor abstraction, which means that any search strategy is compatible with any predictor constellation. Therefore, common search procedures like beam search do not need to be reimplemented for every new model or toolkit.

Secs. 2 to 4 describe central concepts in SGNMT like predictors and decoders briefly and outline some common use cases. Sec. 5 shows that the SGNMT software architecture has proven to be very well suited for our research as new directions can be quickly prototyped, and new NMT toolkits can be introduced without breaking old code. Sec. 6 and Sec. 7 discuss the benefits of SGNMT in teaching and industry, respectively.

[3]Making all models of the T2T library (Google, 2017) available to SGNMT took less than 200 lines of code.

[4]For example, the neural language modeling software NPLM (Vaswani et al., 2013) is written in C++, but can be accessed in SGNMT via its Python interface.

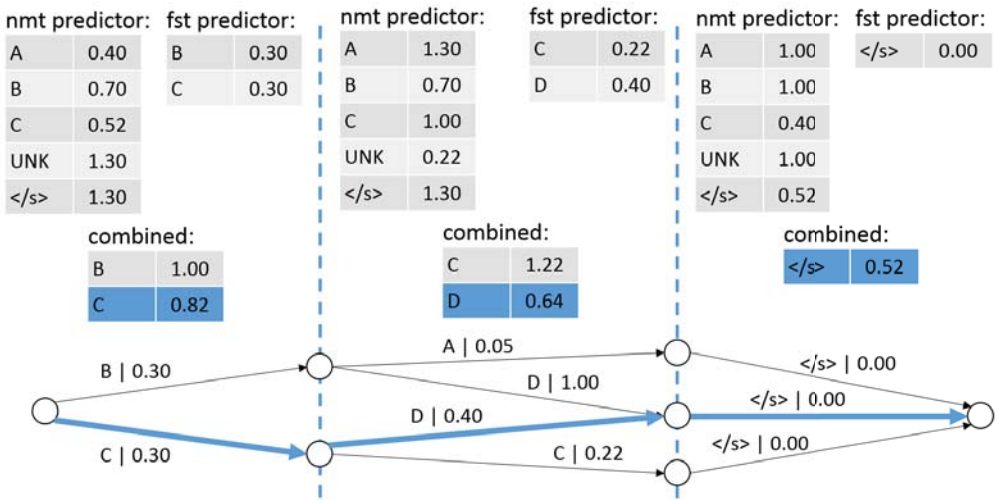

Figure 2: Greedy decoding with the predictor constellation *nmt*,*fst* for lattice rescoring.

2 The Predictor Interface

Predictors in SGNMT provide a uniform interface for models and constraints. Since predictors are decoupled from each other, any predictor can be combined with any other predictor in a linear model. One predictor usually has a single responsibility as it represents a single model or type of constraint. Predictors need to implement the following methods:

- `initialize(src_sentence)` Initialize the predictor state using the source sentence.
- `get_state()` Get the internal predictor state.
- `set_state(state)` Set the internal predictor state.
- `predict_next()` Given the internal predictor state, produce the posterior over target tokens for the next position.
- `consume(token)` Update the internal predictor state by adding `token` to the current history.

The structure of the predictor state and the implementations of these methods differ substantially between predictors. Stahlberg et al. (2017b) provide a full list of available predictors. Fig. 2 illustrates how the *fst* and the *nmt* predictors work together to carry out (greedy) lattice rescoring with an NMT model. The `predict_next()` method of the *nmt* predictor produces a distribution over the complete NMT vocabulary $\{\text{A}, \text{B}, \text{C}, \text{UNK}, \text{</s>}\}$ at each time step in form of negative log probabilities. The *fst* predictor returns the scores of symbols with an outgoing arc from the current node in the FST in `predict_next()`. The linear combination of both scores is used to select the next word, which is then fed back to the predictors via `consume()`. Words outside a predictor vocabulary are automatically matched with the UNK score. For instance, 'D' in Fig. 2 is matched with the NMT 'UNK' token. Pseudo-code for the predictors and the decoder is listed in Figs. 3 and 4, respectively.

```
class NMTPredictor(Predictor):
  def initialize(src_sentence):
    enc_states = enc_computation_graph(
        src_sentence)
    dec_input = [BOS]
  def predict_next():
    scores, dec_state = \
        dec_computation_graph(
            dec_input, enc_states)
    return scores
  def consume(word):
    dec_input = word
  def get_state():
    return dec_state, dec_input
  def set_state(state):
    dec_state, dec_input = state
```

(a) The *nmt* predictor

```
class FSTPredictor(Predictor):
  def initialize(src_sentence):
    Load FST file
    cur_node = start_node

  def predict_next():
    return outgoing_arcs(cur_node)

  def consume(word):
    cur_node = cur_node.arcs[word]

  def get_state():
    return cur_node

  def set_state(state):
    cur_node = state
```

(b) The *fst* predictor

Figure 3: Pseudo-code predictor implementations

```
class GreedyDecoder(Decoder):
  def decode(src_sentence):
    initialize_predictors(src_sentence)
    trgt_sentence = []
    trgt_word = None
    while trgt_word != EOS:
      trgt_word = argmin(combine(predictors.predict_next()))
      trgt_sentence.append(trgt_word)
      predictors.consume(trgt_word)
    return trgt_sentence
```

Figure 4: Pseudo-code implementation of greedy decoding

3 Search Strategies

Search strategies, called *Decoders* in SGNMT, search over the space spanned by the predictors. We use different decoders for different predictor constellations, e.g. heuristic search for bag-of-words problems (Hasler et al., 2017), or beam search for NMT. SGNMT can also be used to analyze search errors. Tab. 1 compares five different search configurations for SMT lattice rescoring with a Transformer model (Vaswani et al., 2017) on a subset[5] of the Japanese-English Kyoto Free Translation Task (KFTT) test set (Neubig, 2011). Following Stahlberg et al. (2016) we measure time complexity in number of node expansions. Our depth-first search algorithm stops when a partial hypothesis score is worse than the current best complete hypothesis score (admissible pruning), but it is guaranteed to return the global best model score. Beam search yields a significant amount of search errors, even with a large beam of 20. Interestingly, a reduction in search errors does not benefit the BLEU score in this setting.

[5]SMT lattices are lightly pruned by removing paths whose weight is more than five times the weight of the shortest path. For the experiments in Tab. 1 we removed very long sentences from the original test set to keep the runtime under control. Lattices have 271 nodes and 408 arcs on average.

	Average number of node expansions per sentence	Sentences with search errors	BLEU score
Exhaustive enumeration	652.3K	0%	21.7
Depth-first search with admissible pruning	3.0K	0%	21.7
Beam search (beam=20)	250.5	20.3%	21.9
Beam search (beam=4)	64.8	41.9%	21.9
Greedy decoding	18.0	67.9%	22.1

Table 1: BPE-level SMT lattice rescoring with different search strategies. The BLEU score does not benefit from less search errors due to modeling errors.

	Pure NMT	SMT lattice rescoring	MBR-based NMT-SMT hybrid
Theano: Blocks (van Merriënboer et al., 2015)	18.4	18.9	19.0
TensorFlow: seq2seq tutorial[6]	17.5	19.3	19.2
TensorFlow: NMT tutorial[7]	18.8	19.1	20.0
TensorFlow: T2T Transformer (Google, 2017)	21.7	19.3	22.5

Table 2: BLEU scores of SGNMT with different NMT back ends on the complete KFTT test set (Neubig, 2011) computed with `multi-bleu.pl`. All neural systems are BPE-based (Sennrich et al., 2016) with vocabulary sizes of 30K. The SMT baseline achieves 18.1 BLEU.

4 Output Formats

SGNMT supports five different output formats.

- `text`: Plain text file with first best translations.
- `nbest`: n-best list of translation hypotheses.
- `sfst`: Lattice generation in OpenFST (Allauzen et al., 2007) format with standard arcs.
- `fst`: Lattices with sparse tuple arcs (Iglesias et al., 2015) which keep predictor scores separate.
- `ngram`: MBR-style n-gram posteriors (Kumar and Byrne, 2004; Tromble et al., 2008) as used by Stahlberg et al. (2017a) for NMT.

5 SGNMT for Research

SGNMT is designed for environments in which implementation time is far more valuable than computation time. This basic design decision is strongly reflected by the software architecture which accepts degradations in runtime in favor of extendibility and flexibility. We designed SGNMT that way because training models and coding usually take the most time in our day-to-day work. Decoding, however, usually takes a small fraction of that time. Therefore, reducing the implementation time has a much larger impact on the overall productivity of our research group than improvements in runtime, especially since decoding can be easily parallelized on multiple machines.

Another benefit of SGNMT's predictor framework is that it enables us to write code independently of any NMT package, and swap the NMT back end with more recent software if

[6] `https://github.com/ehasler/tensorflow`

[7] `https://github.com/tensorflow/nmt`, trained with Tensor2Tensor (Google, 2017)

needed. For example, our previous research work on lattice rescoring (Stahlberg et al., 2016) and MBR-based NMT (Stahlberg et al., 2017a) used the NMT package Blocks (van Merriënboer et al., 2015) which is based on Theano (Bastien et al., 2012). Since both Blocks and Theano have been discontinued, we recently switched to a Tensor2Tensor (Google, 2017) back end based on TensorFlow (Abadi et al., 2016). Without reimplementation, we could validate that MBR-based NMT holds up even under a much stronger NMT model, the Transformer model (Vaswani et al., 2017). Tab. 2 compares the performance of lattice rescoring and MBR-based combination across four different NMT implementations using SGNMT.

6 SGNMT for Teaching

SGNMT is being used for teaching at the University of Cambridge in course work and student research projects. In the 2015-16 academic year, two students on the Cambridge MPhil in Machine Learning, Speech and Language Technology used SGNMT for their dissertation projects. The first project involved using SGNMT with OpenFST (Allauzen et al., 2007) for applying subword models in SMT (Gao, 2016). The second project developed automatic music composition by LSTMs where WFSAs were used to define the space of allowable chord progressions in 'Bach' chorales (Tomczak, 2016). The LSTM provides the 'creativity' and the WFSA enforces constraints that the chorales must obey. This year, SGNMT provides the decoder for a student project about simultaneous neural machine translation.

SGNMT is also part of two practicals for MPhil students at Cambridge.[8] The first practical applies different kinds of language models to restore the correct casing in a lowercased sentence using FSTs. Since SGNMT has good support for the OpenFST library (Allauzen et al., 2007) and can both read and write FSTs, it is used to integrate neural models such as RNN LMs into the exercise. The second practical focuses on decoding strategies for NMT and explores the synergies of word- and subword-based models and the potential of combining SMT and NMT.

7 SGNMT in the Industry

SDL Research continuously balances the research and development of neural machine translation with a focus on bringing state-of-the-art MT products to the market[9] while pushing the boundaries of MT technology via innovation and quick experimental research.

In this context, it is highly desirable to use versatile tools that can be easily extended to support and combine new models, allowing for quick and painless experimentation. SDL Research chose SGNMT over all other existing tools for rapid prototyping and assessment of new research avenues. Among other Neural MT innovations, SDL Research used SGNMT to prototype and assess attention-based Neural MT (Bahdanau et al., 2015), Neural MT model shrinking (Stahlberg and Byrne, 2017) and the recent Transformer model (Vaswani et al., 2017). As described in Sec. 5, the Transformer model is trivially supported by the SGNMT decoder through its predictor framework, and is easy to combine with other predictors. It is worth noting that at the time of writing this paper, Transformer ensembles are not natively supported by the Tensor2Tensor decoder (Google, 2017).

Although SDL Research's decoder is homegrown, the SGNMT decoder is still a valuable reference tool for side-by-side comparison between state-of-the-art Neural MT research and the Neural MT product.

[8] http://ucam-smt.github.io/sgnmt/html/kyoto_nmt.html

[9] http://www.sdl.com/software-and-services/translation-software/

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. (2007). OpenFST: A general and efficient weighted finite-state transducer library. In *Implementation and Application of Automata*, pages 11–23. Springer.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *ICLR*, Toulon, France.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., Warde-Farley, D., and Bengio, Y. (2012). Theano: new features and speed improvements. In *NIPS*, South Lake Tahoe, Nevada, USA.

Bertoldi, N., Cattoni, R., Cettolo, M., Farajian, M., Federico, M., Caroselli, D., Mastrostefano, L., Rossi, A., Trombetti, M., Germann, U., et al. (2017). MMT: New open source MT for the translation industry. In *Proceedings of The 20th Annual Conference of the European Association for Machine Translation (EAMT)*.

Chung, J., Cho, K., and Bengio, Y. (2016). A character-level decoder without explicit segmentation for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1693–1703. Association for Computational Linguistics.

Gao, J. (2016). Variable length word encodings for neural translation models. MPhil dissertation, University of Cambridge.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. *ArXiv e-prints*.

Google (2017). Tensor2Tensor: A library for generalized sequence to sequence models. `https://github.com/tensorflow/tensor2tensor`. Accessed: 2017-12-12, version 1.3.1.

Hasler, E., Stahlberg, F., Tomalin, M., de Gispert, A., and Byrne, B. (2017). A comparison of neural models for word ordering. In *Proceedings of the International Natural Language Generation Conference*, Santiago de Compostela, Spain.

Helcl, J. and Libovický, J. (2017). Neural Monkey: An open-source tool for sequence learning. *The Prague Bulletin of Mathematical Linguistics*, pages 5–17.

Hieber, F., Domhan, T., Denkowski, M., Vilar, D., Sokolov, A., Clifton, A., and Post, M. (2017). Sockeye: A toolkit for neural machine translation. *ArXiv e-prints*.

Iglesias, G., de Gispert, A., and Byrne, B. (2015). Transducer disambiguation with sparse topological features. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2275–2280. Association for Computational Linguistics.

Jean, S., Cho, K., Memisevic, R., and Bengio, Y. (2015). On using very large target vocabulary for neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1–10. Association for Computational Linguistics.

Junczys-Dowmunt, M., Dwojak, T., and Hoang, H. (2016). Is neural machine translation ready for deployment? A case study on 30 translation directions. In *Proceedings of the 9th International Workshop on Spoken Language Translation (IWSLT)*, Seattle, WA.

Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. (2017). OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72. Association for Computational Linguistics.

Kumar, S. and Byrne, W. (2004). Minimum Bayes-risk decoding for statistical machine translation. In *HLT-NAACL*, pages 169–176, Boston, MA, USA.

Luong, T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421. Association for Computational Linguistics.

Neubig, G. (2011). The Kyoto free translation task. http://www.phontron.com/kftt.

Neubig, G., Morishita, M., and Nakamura, S. (2015). Neural reranking improves subjective quality of machine translation: NAIST at WAT2015. In *WAT*, Kyoto, Japan.

Sennrich, R., Firat, O., Cho, K., Birch, A., Haddow, B., Hitschler, J., Junczys-Dowmunt, M., Läubli, S., Miceli Barone, A. V., Mokry, J., and Nadejde, M. (2017). Nematus: a toolkit for neural machine translation. In *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 65–68. Association for Computational Linguistics.

Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725. Association for Computational Linguistics.

Stahlberg, F. and Byrne, B. (2017). Unfolding and shrinking neural machine translation ensembles. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1946–1956. Association for Computational Linguistics.

Stahlberg, F., de Gispert, A., Hasler, E., and Byrne, B. (2017a). Neural machine translation by minimising the Bayes-risk with respect to syntactic translation lattices. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 362–368. Association for Computational Linguistics.

Stahlberg, F., Hasler, E., Saunders, D., and Byrne, B. (2017b). SGNMT – A flexible NMT decoding platform for quick prototyping of new models and search strategies. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 25–30. Association for Computational Linguistics. Full documentation available at `http://ucam-smt.github.io/sgnmt/html/`.

Stahlberg, F., Hasler, E., Waite, A., and Byrne, B. (2016). Syntactically guided neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 299–305. Association for Computational Linguistics.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Tomczak, M. (2016). Bachbot. MPhil dissertation, University of Cambridge.

Tromble, R. W., Kumar, S., Och, F., and Macherey, W. (2008). Lattice minimum Bayes-risk decoding for statistical machine translation. In *EMNLP*, pages 620–629, Honolulu, HI, USA.

van Merriënboer, B., Bahdanau, D., Dumoulin, V., Serdyuk, D., Warde-Farley, D., Chorowski, J., and Bengio, Y. (2015). Blocks and fuel: Frameworks for deep learning. *CoRR*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 6000–6010. Curran Associates, Inc.

Vaswani, A., Zhao, Y., Fossum, V., and Chiang, D. (2013). Decoding with large-scale neural language models improves translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1387–1392, Seattle, Washington, USA. Association for Computational Linguistics.

Williams, P., Sennrich, R., Nadejde, M., Huck, M., Hasler, E., and Koehn, P. (2014). Edinburghs syntax-based systems at WMT 2014. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 207–214, Baltimore, Maryland, USA. Association for Computational Linguistics.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.