CytonMT: an Efficient Neural Machine Translation Open-source Toolkit Implemented in C++

Xiaolin Wang Masao Utiyama Eiichiro Sumita
Advanced Translation Research and Development Promotion Center
National Institute of Information and Communications Technology, Japan
{xiaolin.wang,mutiyama,eiichiro.sumita}@nict.go.jp

Abstract

This paper presented an open-source neural machine translation toolkit named CytonMT[1]. The toolkit was built from scratch using C++ and Nvidia's GPU-accelerated libraries. The toolkit featured training efficiency, code simplicity and translation quality. Benchmarks showed that cytonMT accelerated the training speed by 64.5% to 110.8% and achieved a high translation quality only lower than the Google's production engine among the NMT systems of attention-based RNN encoder-decoder.

1 Introduction

Neural Machine Translation (NMT) has made remarkable progress over the past few years (Sutskever et al., 2014; Bahdanau et al., 2014; Wu et al., 2016). Just like Moses (Koehn et al., 2007) did for statistic machine translation (SMT), open-source NMT toolkits contributed greatly to this progress, including but not limited to,

- RNNsearch-LV (Jean et al., 2015)[2]
- Luong-NMT (Luong et al., 2015a)[3]
- DL4MT by Kyunghyun Cho et al.[4]
- BPE-char (Chung et al., 2016)[5]
- Nematus (Sennrich et al., 2017)[6]
- OpenNMT (Klein et al., 2017)[7]
- Seq2seq (Britz et al., 2017)[8]
- ByteNet (Kalchbrenner et al., 2016)[9]
- ConvS2S (Gehring et al., 2017)[10]
- Tensor2Tensor (Vaswani et al., 2017)[11]

These open-source NMT toolkits were undoubtedly excellent software. However, there was a common issue among these toolkits – they were all written in script languages with dependencies on third-party GPU frameworks apart from the manufacturer's libraries (see the table 1).

Using script languages and third-party GPU frameworks was a two-edged sword. On one hand, it greatly reduced the workload of coding neural networks. On the other hand, it also caused two problems as follows,

- The running efficiency might decrease, and profiling and optimization became difficult, because the interpreters of the script languages and the third-party frameworks became additional layers between GPUs and users' programs. As NMT systems typically required days to weeks to train, training efficiency was a paramount concern. Slightly faster training could make the difference between plausible and impossible experiments (Klein et al., 2017).
- The Researchers who were using these NMT toolkits were constrained by the frameworks. Certain unexplored or unusual computations

[1] https://github.com/arthurxlw/cytonMt
[2] https://github.com/sebastien-j/LV_groundhog
[3] https://github.com/lmthang/nmt.hybrid
[4] https://github.com/nyu-dl/dl4mt-tutorial
[5] https://github.com/nyu-dl/dl4mt-cdec
[6] https://github.com/EdinburghNLP/nematus
[7] https://github.com/OpenNMT/OpenNMT-py
[8] https://github.com/google/seq2seq
[9] https://github.com/paarthneekhara/byteNet-tensorflow (unofficial) and others.
[10] https://github.com/facebookresearch/fairseq
[11] https://github.com/tensorflow/tensor2tensor

Toolkit	Language	Framework
RNNsearch-LV	Python	Theano,GroundHog
Luong-NMT	Matlab	Matlab
DL4MT	Python	Theano
BPE-char	Python	Theano
Nematus	Python	Theano
OpenNMT	Lua	Torch
Seq2seq	Python	Tensorflow
ByteNet	Python	Tensorflow
ConvS2S	Lua	Torch
Tensor2Tensor	Python	Tensorflow
CytonMT	C++	–

Table 1: Open-source NMT toolkits

might not be allowed or could not be done efficiently, while these computations could be a key to develop novel neural network techniques.

CytonMT were developed to address this issue, in the hope of providing the community an alternative. The toolkit was written in C++ language which was the genuine official language of NVIDIA – the manufactory of the GPU. This gave the toolkit an advantage on efficiency when compared with other toolkits. In addition, the training procedure in the toolkit was designed and implemented with great care to avoid inefficiency.

Implementing in C++ language also gave the toolkit great flexibility or freedom on programming. The researchers who were interested in the real calculations happening inside neural networks could trace source codes down to kernel functions, matrix operations or NVIDIS's APIs, and then modify the codes freely to test their novel ideas.

The code simplicity of CytonMT was comparable to those NMT toolkits which were implemented in script languages. This owed to an open-source general-purpose neural network library in C++, named CytonLib, which was shipped as part of the source codes. The library defined a simple and friendly pattern for users to build arbitrary network architectures, in the cost of two lines of genuine C++ codes per layer. Building machine translation system upon this library greatly reduced the complexity of coding.

CytonMT achieved high translation quality which was the main purpose of NMT toolkits. The toolkit implemented the popular framework of attention-based RNN encoder-decoder. Among the reported systems of the same architectures, it ranked the second place on the benchmark of

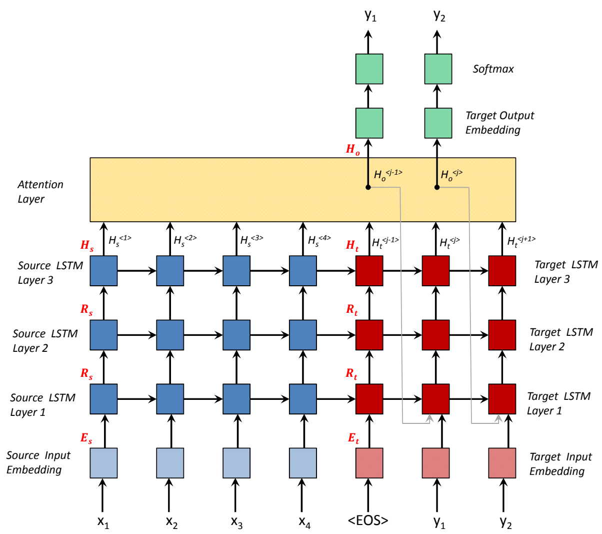

Figure 1: Model Architecture of CytonMT

WMT 2014 English-to-German translation task [12], and only lost to Google' previous industrial engine GPNMT (Wu et al., 2016).

The following of this paper presented details on the method, implementation and benchmark of CytonMT, with a description on future work.

2 Method

The toolkit approached to the problem of machine translation using the attention-based RNN encoder-decoder proposed by Bahdanau et al. (2014) and Luong et al. (2015a). The figure 1 illustrated the method. The conditional probability of a translation given a source sentence was formalized as,

$$\begin{aligned} log\, p(\mathbf{y}|\mathbf{x}) &= \sum_{j=1}^{m} log(p(y_j|H_o^{\langle j \rangle}) \\ &= \sum_{j=1}^{m} \log(\mathrm{softmax}_{y_j}(\tanh(W_o H_o^{\langle j \rangle} + B_o))) \quad (1) \end{aligned}$$

$$H_o^{\langle j \rangle} = \mathcal{F}_{\mathrm{att}}(H_s, H_t^{\langle j \rangle}), \quad (2)$$

where $\mathbf{x}$ was a source sentence; $\mathbf{y}$=$(y_1, \ldots, y_m)$ was a translation; H_s was a source-side top-layer hidden state; $H_t^{\langle j \rangle}$ was a target-side top-layer hidden state; $H_o^{\langle j \rangle}$ was a state output by an attention model $\mathcal{F}_{\mathrm{att}}$; W_o and B_o are the matrix and bias of an output embedding.

The toolkit adopted the multiplicative attention model proposed by Luong et al. (2015a), because it was slightly more efficient than the additive variant proposed by Bahdanau et al. (2014). This issue was addressed in Britz et al. (2017) and Vaswani et al. (2017). The figure 2 illustrated the

[12] http://www.statmt.org/wmt14/translation-task.html

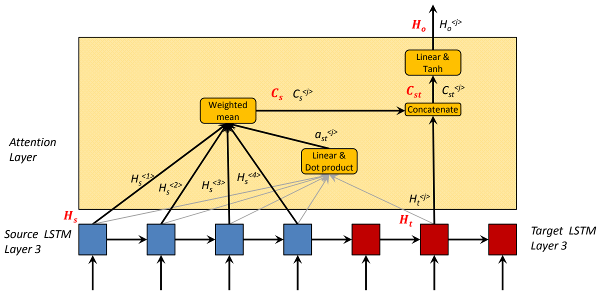

Figure 2: Architecure of Attention Model

model, formulated as ,

$$
\begin{aligned}
a_{st}^{\langle ij \rangle} &= \mathrm{softmax}(\mathcal{F}_{\mathrm{a}}(H_s^{\langle i \rangle}, H_t^{\langle j \rangle})) \\
&= \frac{e^{\mathcal{F}_{\mathrm{a}}(H_s^{\langle i \rangle}, H_t^{\langle j \rangle})}}{\sum_{i=1}^{n} e^{\mathcal{F}_{\mathrm{a}}(H_s^{\langle i \rangle}, H_t^{\langle j \rangle})}}, \qquad (3) \\
\mathcal{F}_{\mathrm{att}}(H_s^{\langle i \rangle}, H_t^{\langle j \rangle}) &= H_s^{\langle i \rangle \top} W_{\mathrm{a}} H_t^{\langle j \rangle}, \qquad (4) \\
C_s^{\langle j \rangle} &= \sum_{i=1}^{n} a_{st}^{\langle ij \rangle} H_s^{\langle i \rangle}, \qquad (5) \\
C_{st}^{\langle j \rangle} &= [C_s; H_t^{\langle j \rangle}], \qquad (6) \\
H_o^{\langle j \rangle} &= \tanh(W_c C_{st}^{\langle j \rangle}), \qquad (7)
\end{aligned}
$$

where $\mathcal{F}_{\mathrm{a}}$ was a scoring function for alignment; W_{a} was a matrix for linearly mapping target-side hidden states into a space which was comparable to the source-side; $a_{st}^{\langle ij \rangle}$ was an alignment coefficient; $C_s^{\langle j \rangle}$ was a source-side context; $C_{st}^{\langle j \rangle}$ was a context derived from both sides.

3 Implementation

The toolkit consisted of a general purpose neural network library written in C++, and a neural machine translation system built upon the library. The neural network library designed a simple and friendly C++ pattern for users to create arbitrary architectures. Users only needed to inherit a base class called *Network*, declare each component as data members, and write down two lines of codes per component in an initialization function. For example, the complete code of the attention network formulated by the equations 3 to 7 was presented in the figure 3. This piece of code fulfilled the task of building a neural network as follows,

- The class of *Variable* stored numeric values and gradients. Through passing the pointer of *Variable* around, each component were connected.

```
class Attention: public Network
{
 DuplicateLayer dupHt;    // declare components
 LinearLayer linearHt;
 MultiplyHsHt multiplyHsHt;
 SoftmaxLayer softmax;
 WeightedHs weightedHs;
 Concatenate concateCsHt;
 LinearLayer linearCst;
 ActivationLayer actCst;

 Variable* init(LinearLayer* linHt,
    LinearLayer* linCst, Variable* hs,
    Variable* ht)
 {
  Variable* tx;
  tx=dupHt.init(ht);            // make two copies
  layers.push_back(&dupHt);

  tx=linearHt.init(linHt, tx);            // WsHt
  layers.push_back(&linearHt);

  tx=multiplyHsHt.init(hs, tx);         // HsWsHt
  layers.push_back(&multiplyHsHt);

  tx=softmax.init(tx);                   // F_att
  layers.push_back(&softmax);

  tx=weightedHs.init(hs, tx);               // Cs
  layers.push_back(&weightedHs);

  tx=concateCsHt.init(tx, &dupHt.y1);       // Cst
  layers.push_back(&concateCsHt);

  tx=linearCst.init(linCst, tx);
  layers.push_back(&linearCst);

  tx=actCst.init(tx, CUDNN_ACTIVATION_TANH);// Ho
  layers.push_back(&actCst);

  return tx; //pointer to result
 }
};
```

Figure 3: Complete Code of Attention Model Formulated by Equations 3 to 7

- The data member of *layers* collected all the components. The base class of *Network* would call the functions *forward*, *backward* and *calculateGradient* of each component to perform the actual computation.

The code of actual computation was organized in the functions *forward*, *backward* and *calculateGradient* of each type of component. The figure 4 presented some examples. Note that these codes were simplified for illustration.

4 Benchmarks

4.1 Settings

The standard WMT 2014 English-to-German training set was chosen as the benchmark dataset (the table 2). To make the experiments easily replicable, the preprocessed dataset from Vaswani et al. (2017)[13] was used. The dataset encoded sentences using byte-pair encoding(Gage,

[13] https://github.com/tensorflow/tensor2tensor

```
void LinearLayer::forward()
{
 cublasXgemm(cublasH, CUBLAS_OP_T, CUBLAS_OP_N,
  dimOutput, num, dimInput,
  &one, w.data, w.ni, x.data, dimInput,
  &zero, y.data, dimOutput)
}

void LinearLayer::backward()
{
 cublasXgemm(cublasH, CUBLAS_OP_N, CUBLAS_OP_N,
  dimInput, num, dimOutput,
  &one, w.data, w.ni, y.grad.data, dimOutput,
  &beta, x.grad.data, dimInput));
}

void LinearLayer::calculateGradient()
{
 cublasXgemm(cublasH, CUBLAS_OP_N, CUBLAS_OP_T,
  dimInput, dimOutput,  num,
  &one, x.data, dimInput, y.grad.data, dimOutput,
  &one, w.grad.data, w.grad.ni));
}

void EmbeddingLayer::forward()
{
 ...
 embedding_kernel<<<grid, blockSize>>>(words,
  firstOccurs, len, dim, stride,
  wholeData, y.data, true);
}
```

Figure 4: Codes of Performing Actual Computation.

1994; Schuster and Nakajima, 2012), which had a shared source-target vocabulary of about 37000 tokens.

The benchmarks were run on an Intel Xeon CPU E5-2630 @ 2.4Ghz and a GPU Quadro M4000 (Maxwell) that had 1664 CUDA cores @ 773 MHz, 2,573 GFLOPS. The software was CentOS 6.8, CUDA 9.1 (driver 387.26), CUDNN 7.0.5, Theano 1.0.1, Tensorflow 1.5.0; the Netmaus, Torch and OpenNMT were the last version in December 2017.

The hyperparameters settings of CytonMT were presented by the table 3, which were chosen as the default settings of CytonMT. The settings provided both fast training and reasonably high translate quality according to our experiments on a variety of translation tasks. Dropout were applied to the hidden states between non-top recurrent layers R_s, R_t and output H_o according to (Wang et al., 2017). Label smoothing estimated the marginalized effect of label-dropout during training, which made models learn to be more unsure (Szegedy et al., 2016). This improved BLEU scores (Vaswani et al., 2017). Length penalty was applied through the formula in (Wu et al., 2016).

Data Set	# Sent.	# Words	
		Source	Target
CommonCrawl	2,399,123	54,572,703	58,869,785
Europarl v7	1,959,829	51,7061,34	54,327,972
News Comment.	223,153	5,689,117	5,660,789
Dev. (tst2013)	3,000	64,807	63,412
Test (tst2014)	3,003	67,617	63,078

Table 2: WMT 2014 English-to-German corpora

Hyperparameter	Value
Embedding/State Size	512
Encoder/Decoder Depth	2
Encoder	Bidirectional
RNN Type	LSTM
Dropout	0.2
Label Smooth.	0.1
Optimizer	SGD
Learning Rate	1.0
Learning Rate Decay	0.7
Beam Search Size	10
Length Penalty	0.6

Table 3: Hyperparameter Settings

4.2 Comparison on Training Speed

Three baseline toolkits and CytonMT were run to train models using the settings of hyperparameters in the table 3. The number of layers and the size of embeddings and hidden states varied, as larger networks were often used in real-world applications to achieve better accuracy at the cost of running time.

The table 4 presented the training speed of different toolkits measured in source tokens per second. The results showed that the training speed of CytonMT is much higher than all the baselines. OpenNMT is was the fastest baseline, while CytonMT achieved a speed up versus OpenNMT by 64.5% to 110.8%. Moreover, CytonMT showed a consistent tendency to speed up more on larger networks.

Embed./State Size	512	512	1024	1024
Enc./ Dec. Layers	2	4	2	4
Nematus	1875	1190	952	604
OpenNMT	2872	2038	1356	904
Seq2Seq	1618	1227	854	599
CytonMT	**4725**	**3751**	**2571**	**1906**
speedup $\geqslant$	64.5%	84.1%	89.6%	110.8%

Table 4: Training Speed Measured in Source Tokens per Second.

System	Open Src.	BLEU
Nematus(Klein,2017)	$\surd$	18.25
OpenNMT(Klein,2017)	$\surd$	19.34
RNNsearch-LV(Jean,2015)	$\surd$	19.4
Deep-Att(Zhou,2016)		20.6
Luong-NMT(Luong,2015)	$\surd$	20.9
BPE-Char(Chung,2016)	$\surd$	21.5
Seq2seq(Britz, 2017)	$\surd$	22.19
CytonMT	$\surd$	**22.67** (+0.48)
GNMT (Wu, 2015)		24.61

Table 5: Comparing BLEU with Public Records.

4.3 Comparison on Translation Quality

The table 5 compared the BLEU of CytonMT with reported results from the systems that shared the same architecture of attention-based RNN encoder-decoder. To be comparable to previous work (Sutskever et al., 2014; Luong et al., 2015b; Wu et al., 2016; Zhou et al., 2016) BLEU was calculated on cased, tokenized text using multi-bleu.pl from the public implementation of Moses [14].

CytonMT was run on the standard training set with the hyperparameters settings in the table 3. The training scheme was monitoring the cross entropy of the development set every one-12th epoch. If the development cross entropy had not decreased by $max(0.01 \times learning\_rate, 0.001)$ in 12 times of evaluations (one complete epoch), learning rate decayed by 0.7 and training restarted from the best model in the history. The training was terminated by hand after 28 epochs when no improvement was made by restarting twice, indicating the development cross entropy was unlikely to decrease.

The table 5 showed than CytonMT achieved the highest BLEU points among all the open-source toolkits. CytonMT was only outperformed only Google's production system (Wu et al., 2016), which was very much larger in scale and demanding much power hardware. Note that the start-of-the-art score on this benchmark was recently pushed forward by novel network architectures such as Gehring et al. (2017); Vaswani et al. (2017); Shazeer et al. (2017)

5 Conclusion

This paper introduced CytonMT – an open-source NMT toolkit, built from scratch using C++ and Nvidia's GPU-accelerated libraries. CytonMT sped up training by more than 64.5%, and achieved the highest BLEU point on the benchmark of WMT2014 among the open-source NMT toolkits sharing the same architecture of attention-based RNN encoder-decoder. The source code of CytonMT was also simple because of CytonLib – an open-source general purpose neural network library – contained in the toolkit. Therefore, CytonMT was an attractive alternative for the research community. We open-sourced this toolkit in the hope of benefiting the community and promoting the field like Moses and many other excellent toolkits did. We look forward to hearing feedback from the community.

The future work of CytonMT will be continued in two directions. One direction is to further optimize the code for NVIDA GPU hardware, such supporting multi-GPU. The problem we used to have was that GPU hardware proceeded very fast in the last few years. For example, GPU microarchitectures evolved twice during the development of CytonMT, from Maxwell to Pascale, and then to Volta. Therefore, we have not explored cutting-edge GPU techniques as the coding effort may be outdated quickly. Multi-GPU machines are common now, so we plan to enable it in CytonMT.

The other direction is to support latest NMT architectures such ConvS2S (Gehring et al., 2017) and Transformer (Vaswani et al., 2017). In these architectures, recurrent structures are replaced by convolution or attention structures. The high performance they achieved indicates that the new structures suit the translation task better. We plan to extend the toolkit to support these new architectures in the near future.

[14]https://github.com/moses-smt/mosesdecoder

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *Proceedings of the 3rd International Conference on Learning Representations*. pages 1–15.

Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. 2017. Massive exploration of neural machine translation architectures. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Copenhagen, Denmark, pages 1442–1451.

Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio. 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)*. Association for Computational Linguistics, Berlin, Germany, pages 1693–1703.

Philip Gage. 1994. A new algorithm for data compression. *The C Users Journal* 12(2):23–38.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional Sequence to Sequence Learning. *ArXiv e-prints* .

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 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)*. Association for Computational Linguistics, Beijing, China, pages 1–10.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aäron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. *CoRR* abs/1610.10099.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*. Association for Computational Linguistics, Vancouver, Canada, pages 67–72.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics on Interactive Poster and Demonstration Sessions*. Association for Computational Linguistics, pages 177–180.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015a. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 1412–1421.

Thang Luong, Ilya Sutskever, Quoc Le, Oriol Vinyals, and Wojciech Zaremba. 2015b. Addressing the rare word problem in 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)*. Association for Computational Linguistics, Beijing, China, pages 11–19.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, pages 5149–5152.

Rico Sennrich, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel L"aubli, Antonio Valerio Miceli Barone, Jozef Mokry, and Maria Nadejde. 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*. Association for Computational Linguistics, Valencia, Spain, pages 65–68.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *CoRR* abs/1701.06538.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. pages 3104–3112.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. *Proceedings of the Conference on Computer Vision and Pattern Recognition* pages 2818–2826.

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 I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., pages 6000–6010.

Xiaolin Wang, Masao Utiyama, and Eiichiro Sumita. 2017. Empirical study of dropout scheme for neural machine translation. In *Proceedings of the 16th Machine Translation Summit*. pages 1–15.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* .

Jie Zhou, Ying Cao, Xuguang Wang, Peng Li, and Wei Xu. 2016. Deep recurrent models with fast-forward connections for neural machine translation. *Transactions of the Association for Computational Linguistics* 4:371–383.