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 presents an open-source neural machine translation toolkit named CytonMT[1]. The toolkit is built from scratch only using C++ and NVIDIA's GPU-accelerated libraries. The toolkit features training efficiency, code simplicity and translation quality. Benchmarks show that CytonMT accelerates the training speed by 64.5% to 110.8% on neural networks of various sizes, and achieves competitive translation quality.

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) does for statistic machine translation (SMT), open-source NMT toolkits contribute 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]
- Marian (Junczys-Dowmunt et al., 2018)[12]

These open-source NMT toolkits are undoubtedly excellent software. However, there is a common issue – they are all written in script languages with dependencies on third-party GPU platforms (see Table 1) except Marian which is developed simultaneously with our toolkit.

Using script languages and third-party GPU platforms is a two-edged sword. On one hand, it greatly reduces the workload of coding neural networks. On the other hand, it also causes two problems as follows,

- The running efficiency drops, and profiling and optimization also become difficult, as the direct access to GPUs is blocked by the language interpreters or the platforms. NMT systems typically require days or weeks to train, so training efficiency is a paramount concern. Slightly faster training can make the difference between plausible and impossible experiments (Klein et al., 2017).
- The researchers using these toolkits may be constrained by the platforms. Unexplored computations or operations may become disallowed or unnecessarily inefficient on a third-party platform, which lowers the chances of developing novel neural network techniques.

[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
[12] https://github.com/marian-nmt/marian

Toolkit	Language	Platform
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
Marian	C++	–
CytonMT	C++	–

Table 1: Languages and Platforms of Open-source NMT toolkits

CytonMT is developed to address this issue, in hopes of providing the community an attractive alternative. The toolkit is written in C++ which is the genuine official language of NVIDIA – the manufacturer of the most widely-used GPU hardware. This gives the toolkit an advantage on efficiency when compared with other toolkits.

Implementing in C++ also gives CytonMT great flexibility and freedom on coding. The researchers who are interested in the real calculations inside neural networks can trace source codes down to kernel functions, matrix operations or NVIDIA's APIs, and then modify them freely to test their novel ideas.

The code simplicity of CytonMT is comparable to those NMT toolkits implemented in script languages. This owes to an open-source general-purpose neural network library in C++, named CytonLib, which is shipped as part of the source code. The library defines a simple and friendly pattern for users to build arbitrary network architectures in the cost of two lines of genuine C++ code per layer.

CytonMT achieves competitive translation quality, which is the main purpose of NMT toolkits. It implements the popular framework of attention-based RNN encoder-decoder. Among the reported systems of the same architecture, it ranks at top positions on the benchmarks of both WMT14 and WMT17 English-to-German tasks.

The following of this paper presented the details of CytonMT from the aspects of method, implementation, benchmark, and future works.

2 Method

The toolkit approaches 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 illustrates the architecture. The conditional probability of a translation given a source sentence is formulated as,

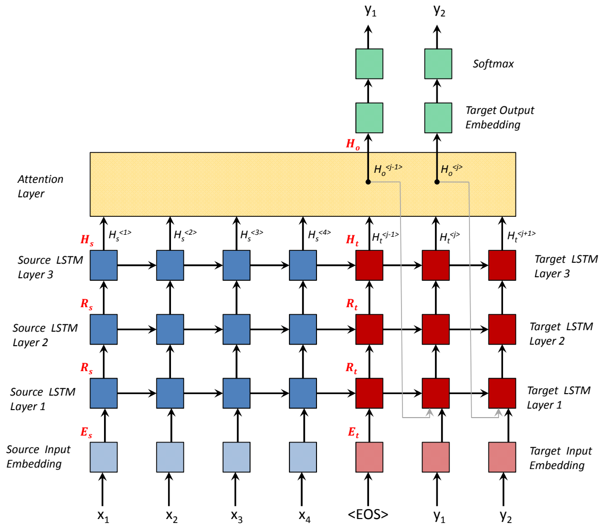

Figure 1: Model Architecture of CytonMT

$$
\begin{aligned}
log\, p(\mathbf{y}|\mathbf{x}) &= \sum_{j=1}^{m} log(p(y_j|H_o^{\langle j\rangle}) \\
&= \sum_{j=1}^{m} \log(\mathrm{softmax}_{y_j}(\tanh(W_o H_o^{\langle j\rangle} + B_o))) \quad (1) \\
H_o^{\langle j\rangle} &= \mathcal{F}_{\mathrm{att}}(H_s, H_t^{\langle j\rangle}), \quad (2)
\end{aligned}
$$

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

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

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

where $\mathcal{F}_{\mathrm{a}}$ is a scoring function for alignment; W_{a} is a matrix for linearly mapping target-side hidden

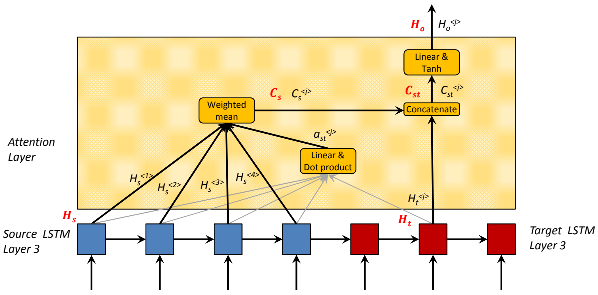

Figure 2: Architecure of Attention Model

states into a space comparable to the source-side; $a_{st}^{\langle ij \rangle}$ is an alignment coefficient; $C_s^{\langle j \rangle}$ is a source-side context; $C_{st}^{\langle j \rangle}$ is a context derived from both sides.

3 Implementation

The toolkit consists of a general purpose neural network library, and a neural machine translation system built upon the library. The neural network library defines a class named *Network* to facilitate the construction of arbitrary neural networks. Users only need to inherit the class, declare components 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 is presented in the figure 3. This piece of code fulfills the task of building a neural network as follows,

- The class of *Variable* stores numeric values and gradients. Through passing the pointers of *Variable* around, component are connected together.

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

The codes of actual computation are organized in the functions *forward*, *backward* and *calculateGradient* for each type of component. The figure 4 presents some examples. Note that these codes have been slightly simplified for illustration.

```
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);             // WaHt
  layers.push_back(&linearHt);

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

  tx=softmax.init(tx);                      // ast
  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);          // WcCst
  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

```
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.

4 Benchmarks

4.1 Settings

CytonMT is tested on the widely-used benchmarks of the WMT14 and WMT17 English-to-German tasks (Bojar et al., 2017) (Table 2). Both datasets are processed and converted using byte-pair encoding(Gage, 1994; Schuster and Nakajima, 2012) with a shared source-target vocabulary of about 37000 tokens. The WMT14 corpora are processed by the scripts from Vaswani et al. (2017)[13]. The WMT17 corpora are processed by the scripts from Junczys-Dowmunt et al. (2018)[14], which includes 10 million back-translated sentence pairs for training.

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 is CentOS 6.8, CUDA 9.1 (driver 387.26), CUDNN 7.0.5, Theano 1.0.1, Tensorflow 1.5.0. Netmaus, Torch and OpenNMT are the latest version in December 2017. Marian is the last version in May 2018.

CytonMT is run with the hyperparameters settings presented by Table 3 unless stated otherwise. The settings provide both fast training and competitive translate quality according to our experiments on a variety of translation tasks. Dropout is 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 estimates the marginalized effect of label-dropout during training, which makes models learn to be more unsure (Szegedy et al., 2016). This improved BLEU scores (Vaswani et al., 2017). Length penalty is applied using the formula in (Wu et al., 2016).

4.2 Comparison on Training Speed

Four baseline toolkits and CytonMT train models using the settings of hyperparameters in Table 3. The number of layers and the size of embeddings and hidden states varies, as large networks are often used in real-world applications to achieve higher accuracy at the cost of more running time.

Table 4 presents the training speed of different toolkits measured in source tokens per second. The results show that the training speed of CytonMT is much higher than the baselines. OpenNMT is the fastest baseline, while CytonMT achieves a speed up versus it by 64.5% to 110.8%. Moreover, CytonMT shows a consistent tendency to speed up more on larger networks.

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

[14]https://github.com/marian-nmt/marian-examples/tree/master/wmt2017-uedin

Data Set	# Sent.	# Words Source	# Words Target
WMT14			
Train.(standard)	4,500,966	113,548,249	107,259,529
Dev. (tst2013)	3,000	64,807	63,412
Test (tst2014)	3,003	67,617	63,078
WMT17			
Train.(standard)	4,590,101	118,768,285	112,009,072
Train.(back trans.)	10,000,000	190,611,668	149,198,444
Dev. (tst2016)	2,999	64,513	62,362
Test (tst2017)	3,004	64,776	60,963

Table 2: WMT English-to-German corpora

Hyperparameter	**Value**
Embedding Size	512
Hidden 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

4.3 Comparison on Translation Quality

Table 5 compares the BLEU of CytonMT with the reported results from the systems of the same architecture (attention-based RNN encoder-decoder). BLEU is calculated on cased, tokenized text to be comparable to previous work (Sutskever et al., 2014; Luong et al., 2015b; Wu et al., 2016; Zhou et al., 2016).

The settings of CytonMT on WMT14 follows Table 3, while the settings on WMT17 adopt a depth of 3 and a hidden state size of 1024 as the training set is three times larger. The cross

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
Marian	2630	1832	1120	688
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
WMT14		
Nematus(Klein,2017)	√	18.25
OpenNMT(Klein,2017)	√	19.34
RNNsearch-LV(Jean,2015)	√	19.4
Deep-Att(Zhou,2016)		20.6
Luong-NMT(Luong,2015)	√	20.9
BPE-Char(Chung,2016)	√	21.5
Seq2seq(Britz, 2017)	√	22.19
CytonMT	√	**22.67**
GNMT (Wu, 2015)		24.61
WMT17		
Nematus(Sennrich,2017)	√	27.5
CytonMT	√	**27.63**
Marian(Junczys,2018)	√	27.7

Table 5: Comparing BLEU with Public Records.

entropy of the development set is monitored every $\frac{1}{12}$ epoch on WMT14 and every $\frac{1}{36}$ epoch on WMT17, approximately 400K sentence pairs. If the entropy has not decreased by $max(0.01 \times learning\_rate, 0.001)$ in 12 times, learning rate decays by 0.7 and the training restarts from the previous best model. The whole training procedure terminates when no improvement is made during two neighboring decays of learning rate. The actual training took 28 epochs on WMT14 and 12 epochs on WMT17.

Table 5 shows that CytonMT achieves the competitive BLEU points on both benchmarks. On WMT14, it is only outperformed by Google's production system (Wu et al., 2016), which is very much larger in scale and much more demanding on hardware. On WMT17, it achieves the same level of performance with Marian, which is high among the entries of WMT17 for a single system. Note that the start-of-the-art scores on these benchmarks have been recently pushed forward by novel network architectures such as Gehring et al. (2017), Vaswani et al. (2017) and Shazeer et al. (2017)

5 Conclusion

This paper introduces CytonMT – an open-source NMT toolkit – built from scratch only using C++ and NVIDIA's GPU-accelerated libraries. CytonMT speeds up training by more than 64.5%, and achieves competitive BLEU points on WMT14 and WMT17 corpora. The source code of CytonMT is simple because of CytonLib – an open-source general purpose neural network library – contained in the toolkit. Therefore, CytonMT is an attractive alternative for the research community. We open-source this toolkit in hopes of benefiting the community and promoting the field. 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 GPUs, such supporting multi-GPU. The problem we used to have is that GPUs proceed very fast in the last few years. For example, the microarchitectures of NVIDIA GPUs evolve 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 support them.

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. Their high performance indicates that the new structures suit the translation task better, so we also plan to support them in the future.

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.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Shujian Huang, Matthias Huck, Philipp Koehn, Qun Liu, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Raphael Rubino, Lucia Specia, and Marco Turchi. 2017. Findings of the 2017 conference on machine translation (wmt17). In *Proceedings of the Second Conference on Machine Translation, Volume 2: Shared Task Papers*, pages 169–214, Copenhagen, Denmark. Association for Computational Linguistics.

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*, pages 1442–1451, Copenhagen, Denmark. Association for Computational Linguistics.

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)*, pages 1693–1703, Berlin, Germany. Association for Computational Linguistics.

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)*, pages 1–10, Beijing, China. Association for Computational Linguistics.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, Andr F. T. Martins, and Alexandra Birch. 2018. Marian: Fast neural machine translation in c++. *arXiv preprint arXiv:1804.00344*.

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, System Demonstrations*, pages 67–72, Vancouver, Canada. Association for Computational Linguistics.

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 ACL, Interactive Poster and Demonstration Sessions*, pages 177–180. Association for Computational Linguistics.

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*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

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)*, pages 11–19, Beijing, China. Association for Computational Linguistics.

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

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*, pages 65–68, Valencia, Spain. Association for Computational Linguistics.

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*, pages 6000–6010. Curran Associates, Inc.

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.