The DeepMind Chinese–English Document Translation System at WMT2020

Lei Yu[*], Laurent Sartran[*], Po-Sen Huang[*], Wojciech Stokowiec[*], Domenic Donato[*]
Srivatsan Srinivasan[*], Alek Andreev[*], Wang Ling[*], Soňa Mokrá, Agustin Dal Lago
Yotam Doron, Susannah Young, Phil Blunsom, Chris Dyer[*]
DeepMind
{leiyu, lsartran, posenhuang, wstokowiec, domenicd,
vatsan, alekandreev, lingwang, sonka, agudallago,
ydoron, susannahy, pblunsom, cdyer}@google.com

Abstract

This paper describes the DeepMind submission to the Chinese→English constrained data track of the WMT2020 Shared Task on News Translation. The submission employs a noisy channel factorization as the backbone of a document translation system. This approach allows the flexible combination of a number of independent component models which are further augmented with back-translation, distillation, fine-tuning with in-domain data, Monte-Carlo Tree Search decoding, and improved uncertainty estimation. In order to address persistent issues with the premature truncation of long sequences we included specialized length models and sentence segmentation techniques. Our final system provides a 9.9 BLEU points improvement over a baseline Transformer on our test set (newstest 2019).

1 Introduction

The WMT2020 Shared Task on translating news data from Chinese into English provides a challenging test for machine translation systems and an ideal domain for researchers to evaluate new techniques. The DeepMind submission to the constrained data track is based on the modular noisy channel document translation architecture advocated by Yu et al. (2020). In this formulation, the posterior probability of a translation is the product of the unconditional probability of the output document (the language model) and the conditional probability of the translation from the output to source (the channel model). By assuming sentences within a document are independently translated, we can train the channel model using readily available parallel sentences, rather than being reliant on less numerous parallel documents, and the language model on monolingual documents. This modular approach allows the components of the system to be implemented and optimized independently while at inference time, when we reason over the posterior distribution of translations given the source document, conditional dependencies between translations are induced by the language model prior.

The core of our document-level translation architecture is the noisy channel reranker. It requires proposal, channel, and language models, each of which is optimized separately using different techniques and approaches. For the proposal and channel models we use Transformer models (Vaswani et al., 2017) (§4.1) with data augmentation (§4.2), such as back translation (Edunov et al., 2018), distillation (Kim and Rush, 2016; Liu et al., 2016), and forward-translated parallel documents. We further improve these sequence-to-sequence (seq2seq) models by fine-tuning them with in-domain data (§4.3). To improve the robustness of the reranker we apply adversarial training and contrastive learning methods for uncertainty estimation (§4.4). Finally, we include candidate translations generated by Monte-Carlo Tree Search (MCTS) (§B) in order to improve the diversity of the candidate pool for the reranker. Our language models are based on the Transformer-XL architecture (Dai et al., 2019) and optimized with distillation and fine-tuning with in-domain data (§5).

During development, we observed weaknesses in our system's translations for long sentences, largely due to premature truncations. We developed several techniques to mitigate this issue such as sentence segmentation (breaking sentences into logical complete segments) and training specialized models with synthetically constructed long sequences to generate additional proposals for our reranker (§A).

Experiments show that the aforementioned techniques are very effective: our system outperforms the Transformer baseline by 9.9 BLEU points on our test set (newstest2019). Our final system achieves a BLEU score of 35.4 on the

*Equal contribution.

Proceedings of the 5th Conference on Machine Translation (WMT), pages 326–337
Online, November 19–20, 2020.

Chinese→English news test set of WMT2020.

2 Document Translation via Bayes' Rule

Following Yu et al. (2020), we model document translation via Bayes' rule. We define $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_I)$ as the source document with I sentences, and similarly, $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_J)$ as the target document with J sentences, where $\boldsymbol{x}_i$ and $\boldsymbol{y}_j$ denote the ith sentence in the source document and the jth sentence in the target document respectively. We assume that $I = J$.

The translation of a document $\boldsymbol{X}$ is determined by finding the document $\hat{\boldsymbol{Y}}$, where $p(\hat{\boldsymbol{Y}} \mid \boldsymbol{X})$ is maximal.

$$\begin{aligned}\hat{\boldsymbol{Y}} &= \arg\max_{\boldsymbol{Y}} p(\boldsymbol{Y} \mid \boldsymbol{X}) \\ &= \arg\max_{\boldsymbol{Y}} \underbrace{p(\boldsymbol{X} \mid \boldsymbol{Y})}_{\text{channel model}} \times \underbrace{p(\boldsymbol{Y})}_{\text{language model}} . \end{aligned} \quad (1)$$

We further assume that sentences are independently translated, and that the sentences within a document admit a left-to-right factorization according to the chain rule. Therefore, we have

$$\hat{\boldsymbol{Y}} \approx \arg\max_{\boldsymbol{Y}} \prod_{i=1}^{|\boldsymbol{Y}|} p(\boldsymbol{x}_i \mid \boldsymbol{y}_i) \times p(\boldsymbol{y}_i \mid \boldsymbol{Y}_{<i}), \quad (2)$$

where $\boldsymbol{Y}_{<i} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{i-1})$ denotes a document prefix consisting of the first $i-1$ target sentences.

The advantages of this formulation is that during training the translation models can be learned from parallel sentences and monolingual documents which are vastly available in practice unlike parallel documents. During test time, when a source document is observed, conditional dependencies between the translation of the source sentences are created in the posterior.

2.1 Reranking

Because of the global dependencies in the posterior distribution, decoding in the aforementioned document translation model is computationally expensive. Following Yu et al. (2020), we use an auxiliary proposal model $q(\boldsymbol{y} \mid \boldsymbol{x})$, that approximates the posterior distribution using a direct model, to focus our search on promising parts of the output space. We then carry out the reranking process using an iterative beam search, over candidates generated by the proposal model q, to optimize the objective:

$$\begin{aligned}\mathcal{O}(\boldsymbol{X}, \boldsymbol{Y}_{<i}, \boldsymbol{y}_i) = {} & \lambda_1 \log p_{\text{PM}}(\boldsymbol{y}_i \mid \boldsymbol{x}_i) + \\ & \lambda_2 \log p_{\text{AM}}(\boldsymbol{y}_i \mid \boldsymbol{x}_i) + \\ & \lambda_3 \log p_{\text{CM}}(\boldsymbol{x}_i \mid \boldsymbol{y}_i) + \\ & \log p_{\text{LM}}(\boldsymbol{y}_i \mid \boldsymbol{Y}_{<i}) + \\ & \lambda_4 |\boldsymbol{y}_i| + \\ & \mathcal{O}(\boldsymbol{X}, \boldsymbol{Y}_{<i-1}, \boldsymbol{y}_{i-1}), \end{aligned} \quad (3)$$

where p_{PM} is the proposal probabilities model, p_{AM} is the adversarially trained proposal model (§4.4.1), p_{CM} is the channel model (§4.4.2), p_{LM} is the language model (§5.1), and $|\boldsymbol{y}|$ denotes the number of tokens in the sentence $\boldsymbol{y}$. The weights of component models (λs) are hyperparameters to be tuned in experiments.

In practice, we generate for each source sentence $\boldsymbol{x}_i$ in the document $\boldsymbol{X}$, a series of *candidates* $\boldsymbol{y}_i$, using the proposal model q. As all of the terms in the objective, except for p_{LM}, only involve independent target sentences, they can be computed ahead of time in a *scoring* phase. The *scored candidates* are then passed to the reranker, where the language model is evaluated on the successive prefixes explored by the search, and which outputs the final document $\hat{\boldsymbol{Y}}$.

Iterative beam search The algorithm starts with k complete documents using randomly selected candidates for each of the source sentences. We then iterate through every source sentence $\boldsymbol{x}_i$, replacing the randomly picked initial candidate with every available candidate $\boldsymbol{y}_i$. We pick the top k scoring *complete* documents and continue iterating over the document. Unlike traditional beam search used by Yu et al. (2020), we go through every sentence in the document multiple times, until the top 1 translation converges (usually 2 to 4 full iterations). This allows for context from latter sentences in the document to inform the choice of earlier candidates.

Iterative beam search found improvements in the model objective over traditional beam search in 63% of the documents in our test set. Improvements in objective did not translate in a stable improvement in BLEU or METEOR scores (Eqn. 4) – in fact those scores were slightly reduced for a number of documents. Nevertheless, an informal human evaluation of translated documents showed preference for iterative beam search.

Selection of the hyperparameters λ We perform a grid search over the hyperparameters λ to

maximize a metric on the validation set. The metric we use is the following META score, combining corpus-level BLEU, TER, METEOR, and the 0.1-quantile of per-document BLEU, such that:

$$\begin{aligned}(1-\text{META})^4 = \text{TER}\times \\ (1-\text{BLEU})\times \\ (1-\text{METEOR})\times \\ (1-q_{0.1}(\text{BLEU})).\end{aligned} \quad (4)$$

When several configurations of hyperparameters achieve values of META very close to the maximum (within 0.02), we pick the one maximizing BLEU and/or minimizing the L_2 norm of the λs, considered as a vector. This corresponds to an intuitive prior towards giving more weight to the language model.

3 Training Data

To train all of the models used in our system, we made use only of the constrained data provided to shared task participants. In this section, we discuss the preprocessing and normalization techniques we carried out in an attempt to reduce spurious uncertainty in the modeling problem.

Text preprocessing We carried out the following text normalization steps prior to use in any models:

- Text normalization. Unicode canonicalization (NKFD from), replacement of common multiple encoding errors present in training data, standardization of quotation marks into "directional" variants, conversion of any traditional Chinese characters into simplified forms. Replacement of non-American spelling variants with American spellings using the aspell library.[1]
- Segmentation into words. Chinese was segmented into word-like units using the Jieba segmentation tool.[2] Punctuation was split from English words using a purpose-built library. These processes were not completely invertible, but they could be undone with simple rules so as to generate presentation-ready English and Chinese.
- True-casing. Words containing only an initial capital letter that occurred at the start of a sentence were replaced with the capitalized variant that occurred most frequently in other positions of the English monolingual training data. Thus, in the previous sentence the initial token would have been *words* rather than *Words*.

Subword units To encode text into sub-word units, we used the `sentencepiece` tool (Kudo and Richardson, 2018). For seq2seq models (i.e., the channel model and proposal models), we trained the segmentation model on the first 10 million sentences of the parallel training corpus,[3] using joint source and target unigram (Kudo, 2018) subword segmentation algorithm with a target vocabulary of 32K tokens and minimum character coverage of 0.9995, which resulted in 32,768 word pieces.[4] For the language model, we used the English side alone with the same vocabulary size and a character coverage of 1.0.

4 Proposal and Channel Models

The proposal model, used to generate candidate translations, and the scoring models (proposal probability model, adversarially-trained proposal model, channel model), used to compute features for the reranker, are seq2seq models. We describe here how we train and use them.

4.1 Sequence-to-Sequence Model

All our models are based on the Transformer architecture (Vaswani et al., 2017). We increased the inner dimension of the feed-forward network from 4,096 to 8,096 and decreased the model size (d_{model}) from 1,024 to 512, which allowed us to use 12 layers with 16 attention heads each. Additionally, we tied the source and target embedding layers. Following (Vaswani et al., 2018), we applied layer normalization to the input of every sublayer as opposed to its original placement after the element-wise residual addition. We used different dropout values for different components: 0.1 for the multi-head attention, 0.05 in the feed-forward network, and finally 0.3 after the sub-layer. Learning rate schedule and dropout were found using the Batched Gaussian Process Bandits (Desautels et al., 2014) algorithm as implemented by Vizier (Golovin et al., 2017). All other hyperparameters

[1] `http://wordlist.aspell.net/varcon-readme/`

[2] `https://github.com/fxsjy/jieba`

[3] NC followed by CWMT, WikiTitles and UN.

[4] We tried both larger vocabulary sizes and separate vocabularies but neither of these led to an improvement for our system.

were decided upon using grid search. During training, we used a maximum sequence length of 96. For decoding, we used beam search with beam size 6, and set the length penalty alpha to 0.8, and a maximum decoding length of 384. Multi model ensembling was done via softmax output averaging as described in (Freitag et al., 2017).

4.2 Data Augmentation

In this section, we introduce how we augment data based on the given bilingual data and monolingual data. When we train the proposal and channel models, we use all the augmented data along with the original bilingual data.

Back-translation We perform back-translation from monolingual English data using fine-tuned channel models (English→Chinese) with top-k sampling following (Edunov et al., 2018) with $k = 50$ during decoding. We used the same in-domain monolingual data as described in §5.2. We score the back-translated data with fine-tuned proposal (Chinese→English) models, and filter them based on the quantiles of length ratios, sequence log-probability and cross-entropy between one-hot empirical translations and logits from the scorer model. The filtering helped to reduce the size of data from 43.4M to 29.9M paired sentences.

Forward translation to generate synthetic parallel documents We applied a version of our system to monolingual Chinese documents from Gigaword to get synthetic English documents. We only kept documents having between 4 and 25 sentences, we rejected outliers according to their probabilities under the language model, the channel model, and to the overall objective. These were then used to train subsequent versions of the forward (Chinese→English) models.

Data distillation We use knowledge distillation (Kim and Rush, 2016) to do distillation on the original dataset. Specifically, we translate the source-side of the bilingual data using previously trained proposal models (including Right-to-Left (Liu et al., 2016) and Left-to-Right models) and generate distilled candidates. The generated sentences are filtered if BLEU scores are below 30 (Wang et al., 2018; Sun et al., 2019). We then train models on the filtered data along with the original bilingual data and back-translation data. We repeat this process three times using models trained on newly generated data from the previous iteration. We empirically do not find Right-to-Left models significantly differ from Left-to-Right models in performance. Qualitatively we find that distilled data correct few errors in the original bilingual data.

4.3 Fine-tuning

Fine-tuning with in-domain data has been an effective approach for improving translation quality as shown by existing work (Sun et al., 2019; Ng et al., 2019). After training the proposal models with the mix of real and synthetic parallel data, we fine-tuned the models with CWMT and a subset of *newstest2017* and *newstest2018* which were not used for validation.

4.4 Improving Uncertainty Estimation

To improve the robustness of noisy channel reranking, we explore two approaches for improving uncertainty estimation of the seq2seq scoring models.

4.4.1 Adversarially Trained Proposal Models

To simulate different wordings and noises in source and candidate sentences, we follow Cheng et al. (2019) to train the models on noisy adversarial inputs and targets. We use bidirectional language-models to provide the noisy candidates and select the candidates with highest loss (i.e., adversarial source-target inputs). During the training, we optimize the original loss with clean source-target pairs, the language model losses for source and target sides, and the adversarial loss using adversarial source-target inputs. In the final scoring, we use an ensemble of eight adversarially trained models with few differences from Cheng et al. (2019): (a) We explore training with and without the language model losses. Though the models trained without the language model loss generate quite noisy sentences, we empirically find this approach still helps the overall performance. (b) In addition to using the clean hard-labels for the noisy source-target pairs for the adversarial loss as in the original work, we explore a variation using a KL loss between the adversarial source-target logits and the clean source-target logits. We find this variant also improves the overall performance.

4.4.2 Contrastive Channel Models

When scoring candidates, we want the channel models to be sensitive to translation noise (dropped words, permutation, or blanked words) (Edunov et al., 2018). Hence, we develop contrastive training (Yang et al., 2019; Welbl et al., 2020) to train

the models such that it will be more robust in estimating the channel probabilities. Specifically, we use n-gram Transformers (Chelba et al., 2020) with the contrastive loss:

$$\max\left\{\log p(\tilde{\boldsymbol{x}} \mid \boldsymbol{y}) + \eta - \log p(\boldsymbol{x} \mid \boldsymbol{y}), 0\right\}, \quad (5)$$

where $\tilde{\boldsymbol{x}}$ denotes a noisy version (random word deletion, blank, or permutation) of $\boldsymbol{x}$, and $p(\tilde{\boldsymbol{x}} \mid \boldsymbol{y})$ is the perturbed loss term. We ensemble 8 models with a few variants for final channel model scoring. These variants consist of the followings: (1) We use $\eta = \{0.01, 0.001\}$. (2) We use n-gram transformer with $n = 2, 8$. (3) We use models with perturbing source sentences $p(\tilde{\boldsymbol{x}} \mid \boldsymbol{y})$ and models with perturbing target sentences $p(\boldsymbol{x} \mid \tilde{\boldsymbol{y}})$. (4) Instead of using the contrastive loss, we include two models trained to minimize the perturbed loss terms directly. (5) Unlike Yang et al. (2019), where the authors firstly train with the maximum likelihood objective and then finetune with the contrastive loss, we find it empirically works better to train models with linearly increased weights (increasing from 0 to 1 during training) to the contrastive loss (Eq. (5)) along with the original negative log likelihood loss.

4.5 Filtering Candidate Translations

After obtaining candidate translations from strong proposal models, we filter out candidates with length ratio outside of $[e^{-1}, e^{1}]$, or which do not end with end-of-sentence punctuation when the source does, or with more than 4 consecutive identical tokens, or which are excessively compressible, indicating repeated contents, according to the following. We learn a piece-wise linear ordinary least squares model of the zlib-compressed length of true English sentences from their uncompressed length in UTF-8, using the English side of the training data. We then reject candidates the actual compressed length of which is more than 12 standard deviations below their predicted compressed length.

5 Language Model

In this section, we describe the architecture of the language models we used and how we trained them.

5.1 Model

The auto-regressive document language model is a Transformer-XL (Dai et al., 2019), with attention memory length of 512. Following Rae and Razavi (2020), we also used 4-layer blocks of short and long (128-128-128-512) attention memories, capturing short-range correlations in the earlier layers and long-range correlations in the later ones. This led to a 20% speedup of training, and helped the model generalize better to the validation set. We also used knowledge distillation in our Transformer-XL model with a setup similar to Born Again Neural Networks (BANN) (Furlanello et al., 2018), where we regularize the original loss function with term based on the cross-entropy between the new models outputs (student) and the outputs of the original (teacher) model.

Model	Train Data	Fine-tuning	PPL
Transformer-XL	Raw	No	29.4
Transformer-XL	In domain	No	27.4
+ memory + BANN	In domain	No	26.7
+ memory + BANN	In domain	Yes	24.3

Table 1: Language model perplexities per token on the validation set

Let $\mathcal{L}$ denote cross entropy loss function, $\boldsymbol{y}$ one-hot encoded label, $\boldsymbol{s}$ and $\boldsymbol{t}$ outputs of the student and teacher model respectively, then the BANN loss is defined as follows:

$$\mathcal{L}_{\text{BANN}} = \sum_{i=1}^{T} \mathcal{L}(\boldsymbol{y}_i, \boldsymbol{s}_i) \; + \lambda \cdot \mathcal{L}(\boldsymbol{t}_i, \boldsymbol{s}_i). \quad (6)$$

We trained our student network on the loss function in Eqn. 6 and found that $\lambda = 1$ had the best validation perplexity.

5.2 Data

The English data used to train our language models was prepared as described in §3.

In-domain document data for LM training We found that training LMs on a subset of training data that was more closely aligned with the validation set vastly improved the perplexity on the validation and test sets ($\approx 10\%$). To select a well-aligned subset of training data, we ranked the training data according to TF-IDF similarity with each validation document and collected the top 1,000 documents for each validation query together, to form our training data for the LM training. We also tried mixing this sub-sampled in-domain data with the raw data using different weights (essentially equivalent to up-weighting the in-domain data) and found that using purely in-domain data outperformed all other mixing schemes in terms

System	BLEU
Big Transformer	28.1
+ Data augmentation (§4.2)	33.6
+ Fine-tuning (§4.3)	35.8
+ Ensembling (§4.1)	36.6
+ Reranking (§2.1)	37.2
+ Length-targeting improvements (§A)	38.0

Table 2: SacreBLEU scores on newstest2019 Chinese-English.

of held-out perplexities and thus, the in-domain data became our training dataset. As an auxiliary benefit, the model trained on the in-domain dataset (340K iterations) also converged much earlier than the one trained on the raw dataset (500K iterations).

Similar to the sequence model fine-tuning outlined in §4.3, we also fine-tuned our trained language model in order to align the model more closely with the language constructs and domain information in our test data. Table 1 shows the perplexity numbers on the validation set obtained by different train data and model variants described above on the validation dataset.

6 Experiments and Results

We use the original Chinese subset of *newstest2017* and *newstest2018* as our validation set and *newstest2019* as our test set.

The candidate translations for the reranker are generated by 8 ensemble models (6 from each).

Table 2 presents the results of our models on the test set. We report case-sensitive SacreBLEU scores (Post, 2018). Both data augmentation and fine-tuning significantly improve the performance. Ensembling and noisy channel reranking gives about 0.8 and 0.6 BLEU boost, respectively. Finally, our specialized methods for handling long sequences (described in §A) yield a further 0.8 BLEU improvement.

In our final submitted system, we tune the weights of component models and the hyperparameters of sentence segmentation models using a combination of our validation set and test set. For the candidate translations of the reranker, apart from the existing 48 proposals generated by 8 ensemble models, we include additional 48 proposals generated by 8 ensemble models which are fine-tuned with CWMT, newstest2017, newstest2018, and newstest2019. We also include translations generated by MCTS decoding (§B) in our non-primary system. We find that adding a feature marking the length of source sentences longer than 60 words helps the reranker handle long sentences better. We therefore include this feature in addition to proposal probability, adversarial proposal probability, channel probability, language model probability, and length bonus (Eqn. 3).

Our system achieves a 35.4 BLEU score on newstest2020.

7 Conclusion

This paper describes the DeepMind submission to the WMT2020 news Chinese-English translation task. Using the noisy channel model (Yu et al., 2020) as our core document translation system, we optimized its component models using data augmentation, fine-tuning with in-domain data, MCTS decoding (§B), and knowledge distillation. We also addressed premature termination in long sentences by training specialized length expert models and segmenting long sentences into multiple shorter sentences (§A). We have demonstrated the marginal contributions of these methods in our analysis and our final system comprising all these methods outperforms the Transformer baseline by 9.9 BLEU points on newstest2019.

References

Ciprian Chelba, Mia Chen, Ankur Bapna, and Noam Shazeer. 2020. Faster transformer decoding: N-gram masked self-attention. *arXiv preprint arXiv:2001.04589*.

Yong Cheng, Lu Jiang, and Wolfgang Macherey. 2019. Robust neural machine translation with doubly adversarial inputs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4324–4333.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc Viet Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 2978–2988. Association for Computational Linguistics.

Thomas Desautels, Andreas Krause, and Joel Burdick. 2014. Parallelizing exploration-exploitation tradeoffs in gaussian process bandit optimization. *Journal of Machine Learning Research (JMLR)*, 15:40534103.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 489–500.

Markus Freitag, Yaser Al-Onaizan, and Baskaran Sankaran. 2017. Ensemble distillation for neural machine translation. *arXiv preprint arXiv:1702.01802*.

Tommaso Furlanello, Zachary Chase Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. 2018. Born-again neural networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1602–1611. PMLR.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. 2017. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495. ACM.

Mark Hopkins and Jonathan May. 2011. Tuning as ranking. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1352–1362, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas. Association for Computational Linguistics.

Levente Kocsis and Csaba Szepesvri. 2006. Bandit based monte-carlo planning. In *ECML*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Brenden M. Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2879–2888. PMLR.

Lemao Liu, Masao Utiyama, Andrew Finch, and Eiichiro Sumita. 2016. Agreement on target-bidirectional neural machine translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 411–416.

Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook fair's wmt19 news translation task submission. *arXiv preprint arXiv:1907.06616*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers, WMT 2018, Belgium, Brussels, October 31 - November 1, 2018*, pages 186–191.

Jack W. Rae and Ali Razavi. 2020. Do transformers need deep long-range memory? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7524–7529. Association for Computational Linguistics.

Meng Sun, Bojian Jiang, Hao Xiong, Zhongjun He, Hua Wu, and Haifeng Wang. 2019. Baidu neural machine translation systems for WMT19. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 374–381.

Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. 2018. Tensor2Tensor for neural machine translation. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Papers)*, pages 193–199, Boston, MA. Association for Machine Translation in the Americas.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008.

Mingxuan Wang, Li Gong, Wenhuan Zhu, Jun Xie, and Chao Bian. 2018. Tencent neural machine translation systems for wmt18. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 522–527.

Johannes Welbl, Po-Sen Huang, Robert Stanforth, Sven Gowal, Krishnamurthy (Dj) Dvijotham, Martin Szummer, and Pushmeet Kohli. 2020. Towards verified robustness under text deletion interventions. In *International Conference on Learning Representations*.

Nianwen Xue and Yaqin Yang. 2011. Chinese sentence segmentation as comma classification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 631–635, Portland, Oregon, USA. Association for Computational Linguistics.

Zonghan Yang, Yong Cheng, Yang Liu, and Maosong Sun. 2019. Reducing word omission errors in neural machine translation: A contrastive learning approach. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6191–6196.

Lei Yu, Laurent Sartran, Wojciech Stokowiec, Wang Ling, Lingpeng Kong, Phil Blunsom, and Chris Dyer. 2020. Better document-level machine translation with Bayes' rule. *Transactions of Association for Compututational Linguistics*, 8:346–360.

In the appendix, we additionally include the description of our specialized methods for handling long sequences and the MCTS decoding algorithm. The candidate translations generated by MCTS decoding are added to the candidate pool of the noisy channel reranker in our non-primary system.

A Length Considerations

The WMT Chinese evaluation data presents documents as a sequence of segments. These segments are sentence-like units that were delimited in the original article by either unambiguous structural transitions, such as the end of a headline, or unambiguous end-of-sentence punctuation (.!?). However, in some contexts, a complete Chinese sentence may be ended with a comma (Xue and Yang, 2011). This leads to disproportionately more segments in the evaluation data being multi sentence than in the training data, which consists primarily of paired sentences, phrases, and words. Since generalization from short sequences to longer ones is a weakness of neural sequence to sequence models (Lake and Baroni, 2018), to ensure that our candidate pool contains adequate translations of long sentences, we had special handling for long sequences.

A.1 Length Analysis

There is a strong linear relationship between the number of Chinese words in the source segment and the number of English words in the translation (Figure 1 Left). The translations from our initial system were able to match this relationship when the source segment contained less than 60 words. After this point, the translations became too short. Inspection of these translations showed that the primary cause of failure was emitting the EOS token too early. Since the translations were good up to the point of truncation, we focused on methods to prevent early termination. Our final pool consisted of candidates from our original proposal models only for sentences that had less than 80 words and the rest were generated from the techniques outlined here.

A.2 Length Experts

We trained a number of length expert models with a sequence length of 384 tokens. As few ($\leq 1\%$) sentences in the training data were longer than our default sequence length, we used a mixture of real parallel data, synthetic data as described in §**??**, and concatenation of consecutive synthetic sentences to train these length experts. In addition, we also used the original proposal models which were further fine-tuned with long sequences($\geq$ 60 tokens) as additional length experts. These specialized models to handle longer sentences were used to generate proposals for sentences between 60 and 100 words.

A.3 Sentence Segmentation

We found that a lot of the long ($\geq$ 60 words) sentences in our dataset had complete sentences concatenated with commas, semicolons, full-stops, exclamations and question marks. While the latter ones are all conclusive end of sentences, commas are ambiguous as an end of sentence. Hence, we built a comma classifier that distinguished commas that signify end-of-sentence from the normal commas. While training data for this classifier was generated as outlined in (Xue and Yang, 2011), our classification model had feed-forward layers on top of the Transformer encoders (further fine-tuned on this task) that we trained for our translation task. During inference time, we recursively split every sentence on standard end-of-sentence punctuation and then semicolons followed by terminal commas (as determined by the comma classifier), into reasonably sized segments (10-60 words); Very short segments ($<$ 10 words) were merged with their neighboring segments. After this first wave of splits, if long segments ($\geq$ 60 words) still persisted, we further split them recursively on all colons, commas and reverse commas into segments between 40-60 words. Each segment was then translated independently using our translation models. This segmentation procedure was used to generate proposals for all sentences that had more than 60 space-separated words.

Sentence remerging model For each split of a sentence, we obtain a list of candidate translations, and need to combine this. For n splits with k translations each, we have k^n translations in total. Provided n and k are reasonable, we can enumerate these, but it is still too many candidates to present directly to our global reranker, so we need to have a "local" reranker that will select the best k' of these.

To select these, we define a reranking model in terms of our usual features (language model log probability for the remerged sentence to ensure coherence, channel log probabilities for the remerged candidate, sum of the direct translation log probabilities for each segment, and the total length). We

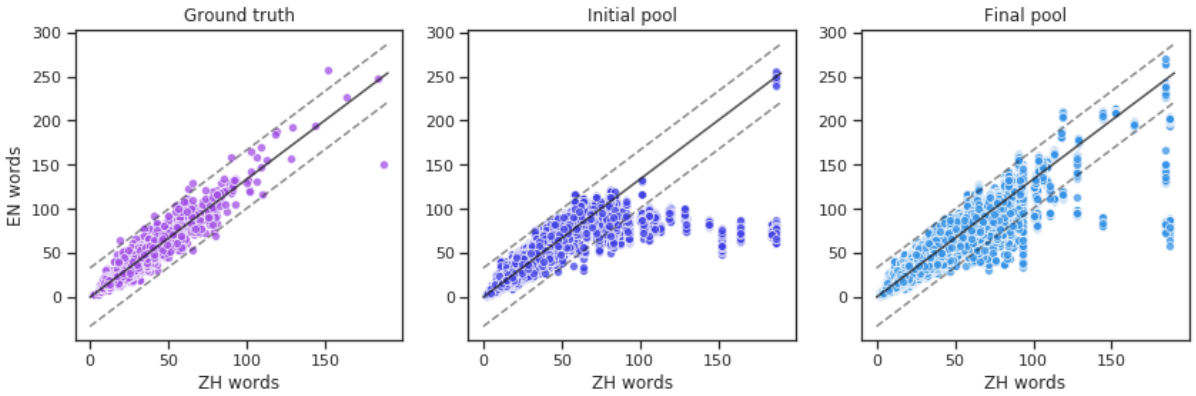

Figure 1: Relationship between the number of white space separated words on normalized text translation pairs. Left is the ground truth for our validation and test datasets. Middle is our initial candidate pool for this data while Right is our pool including candidates from length expert models and our segmentation technique. The black line is the maximum likelihood fit under $|\boldsymbol{y}_i| \sim \mathcal{N}(\beta \cdot |\boldsymbol{x}_i|, \sigma)$ with the dotted grey dashed lines representing the 3σ bounds.

select the remerged candidates k' maximizing an approximate lower bound of the corpus-BLEU.

Approximation to BLEU The weights of the features are learned so as to minimize an approximation to the expected negative log BLEU score. For the reference sentence $\boldsymbol{y}$ and hypothesis $\hat{\boldsymbol{y}}$, the negative expected log BLEU score is defined as:

$$\mathcal{L} = -\mathbb{E}\Big[\min\{0, 1 - |\boldsymbol{y}|/|\hat{\boldsymbol{y}}|\} + \sum_{i=1}^{4} \log c_i(\hat{\boldsymbol{y}}, \boldsymbol{y}) - \log r_i(\boldsymbol{y})\Big],$$

where c_i is a function that counts the clipped i-gram matches against a reference, and r_i counts the i-grams in a reference (Papineni et al., 2002).

Although our model assigns probabilities independently to sentences, BLEU is defined on an entire corpus, and because of the nonlinear functions in BLEU, we cannot compute this expectation tractably. We therefore approximate it by moving the expectations inside the nonlinear functions:

$$\mathcal{L} \approx -\Big[\min\{0, 1 - |\boldsymbol{y}|/\mathbb{E}[|\hat{\boldsymbol{y}}|]\} + \sum_{i=1}^{4} \log \mathbb{E}[c_i(\hat{\boldsymbol{y}}, \boldsymbol{y})] - \log r_i(\boldsymbol{y})\Big].$$

In this approximation, the corpus-level expectations for i-gram counts and the length can be computed tractably using the linearity of expectation. To obtain a learning algorithm, we differentiate this quantity with respect to the weightings of the scoring models and perform gradient descent.

Even with our small number of features, this objective has many local optima and in practice we run the optimizer starting from different positions and find a solution that obtains a high BLEU score and highly weights the language model (during development, we noticed that a higher weight to LM probabilities corresponded to noticeably more fluent translations, even if there was little difference in BLEU).

B MCTS Candidates

The candidate pool generated by the sequence to sequence model optimize the search space that is favorable according to those models and may fail find certain translations that score poorly according to sequence to sequence models, but receive high scores from the noisy channel model. In order to include such translations into the candidate pool, we employ Monte Carlo Tree Search (MCTS), to optimize the noisy channel objective directly. MCTS does not require a partial translation evaluation function, as opposed to Beam Search, making it an appropriate choice for decoding non-factorizable objective functions.

We define the translation environment as a progressive left-to-right language generation process, where each state $s_{\boldsymbol{y}}$ defines a sequence of tokens $\boldsymbol{y}$, and each action a_w appends a word type w at the end of the sequence generating a new state $s_{\boldsymbol{y}'}$. When an action appends the end of sentence token, a terminal state is generated. The reward R for terminal states corresponds to a log-linear combi-

nation of model scores $\phi \in M$ as follows:

$$R(s_{\boldsymbol{y}}) = \exp\left(\sum_{i \in M} \lambda_i \phi_i(\boldsymbol{y})\right), \qquad (7)$$

where λ denotes the weights attributed to each of the models. Our MCTS decoder is composed of two optimization processes running simultaneously. The former aims at finding the optimal state $R(s_{\boldsymbol{y}})$ for each of the sentences in the validation and test sets. The latter maximizes the correlation between the BLEU score on the validation set by optimizing the weights λ.

B.1 Monte Carlo Tree Search with Log Linear Models

MCTS decodes a sentence by growing a search tree. Each node in the tree corresponds to a state but adds additional statistics in order to optimally expand the search tree. Expansion is achieved by applying actions to existing nodes in the tree, generating child nodes. The search process starts with a tree with root node, which corresponds to an empty translation, and gradually expands the tree by append new words to existing nodes in the tree.

Each MCTS iteration performs the following four steps: **selection**, **expansion**, **simulation** and **backpropagation**.

The **selection** step aims at choosing the most likely node in the tree to generate the optimal translation. We employ a standard criteria UCT (Upper Confidence Bounds for Trees (Kocsis and Szepesvri, 2006)), which selects nodes recursively starting from the root according to the following criteria:

$$\text{UCT}(s) = Q(s) + b\sqrt{\frac{2 \ln N(s')}{N(s)}},$$

where s denotes the current node and s' is the parent node. $N(s)$ denotes the number of times s was traversed by the selection process and $Q(s)$ denotes the average reward obtained from s in the $N(s)$ traversals. b is a constant that quantifies the trade-off between exploitation and exploration. We set it to $b = 1$ in our experiments.

In general the average value $Q(s)$ is computed by accumulating the reward $V(s)$ obtained one any traversals containing s, then computing $Q(s) = \frac{V(s)}{N(s)}$. However, as our reward function (Eqn. 7) is concurrently updated by optimizing the weights λ, we store the accumulative individual scores $V_i(s)$ of each of the models ϕ_i. Prior to a MCTS iteration, we update λ and compute $Q(s)$ as follows:

$$Q(s) = \frac{\exp(\sum_{i \in M} \lambda_i V_i(s))}{N}.$$

This allows changes to the weights to be directly reflected in the entire search tree without the recomputation of any tree statistics. As for models M, we trained 9 proposal models and 7 channel models with the architecture defined in §4.1, and 8 language models with the architecture defined in §5.1.

Once a node s is selected, a new child s' is added to the tree in the **expansion** step.

The **simulation** step attempt to compute the expected reward for s'. This is generally accomplished by performing multiple random rollouts starting from state s', where actions are sampled from an uniform distribution until a terminal state is found. These states are then scored according to Eqn. 7. The estimate of the expected reward of s' is computed as the mean of the scores of different rollouts. However, the sparsity underlying natural language generation and the computational complexity of the scoring function makes this practice computationally challenging. Rather than performing multiple rollouts that sample from an uniform distribution at each timestamp, we perform a single rollout that runs greedy decoding starting from the translation prefix defined from state s'. As it is computationally expensive to perform decoding for each MCTS iteration, use a light-weight proposal model trained on the same data, but with a simpler architecture. For this purpose, we reduce the architecture described in §4.1 into a single layer seq2seq layer with 128 hidden units. While this network underfits the data, we found that this trade-off is desirable as it reduces the large dimensionality of the vocabulary to the few examples that are sensible at each prefix leading to a significant speed-up. The quality of the found translation remains unaltered as the reward is still computed with the full models.

Finally, each of the model scores $V_i(s')$ is propagated to all nodes from the root to s' in the **backpropagation** step.

B.2 Pairwise Reranking Optimization

In order to optimize the weights λ in Eqn. 7, we we employ the weight optimization method for log-linear models described in (Hopkins and May, 2011), which allows us to optimize our log-linear

model with respect to the non-differentiable objective function that is BLEU. This method, denominated as PRO, approximates the objective function by training a binary classifier, such that two translations $\boldsymbol{y}$ and $\boldsymbol{y}'$ respect the following equality:

$$g(\boldsymbol{y}) > g(\boldsymbol{y}') \Leftrightarrow \sum_{i \in M} \lambda_i(\phi_i(\boldsymbol{y}) - \phi_i(\boldsymbol{y}')) > 0,$$

where g is the objective function, namely BLEU. Thereby, this requires the generation of pairs $\boldsymbol{y}$ and $\boldsymbol{y}'$, where the LHS property holds. These samples are then used as data to train the model defined in the RHS.

We generate the data by sampling from the MCTS tree for each sentence pair in the development set. As $Q(s)$ is the expected score of a log-linear model with probabilities as components ϕ, we expect that $Q(s)$ is bounded in the $[0, 1]$ interval. Thus, at node s', the probability of sampling child s is given by $\frac{Q(s)}{\sum_{c \in C(s')} Q(c)}$, where $C(s)$ denotes all children of node s'. Once a leaf node is found, if it's terminal we sample its translation, otherwise we sample the translation obtained from its rollout.

Both MCTS search and PRO optimization are executed in parallel, as the former needs optimal weights in order to optimally grow the search tree, and the latter needs the search tree to generate candidates. Thus, at each iteration, the PRO optimizer samples from the most updated version of the tree for each data point in the development set, and updates the set of weights λ, which are then used in the subsequent MCTS iterations.