The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation

Mia Xu Chen [*] **Orhan Firat** [*] **Ankur Bapna** [*]

Melvin Johnson **Wolfgang Macherey** **George Foster** **Llion Jones** **Niki Parmar**

Mike Schuster **Zhifeng Chen** **Yonghui Wu** **Macduff Hughes**

miachen,orhanf,ankurbpn,yonghui@google.com

Google AI

Abstract

The past year has witnessed rapid advances in sequence-to-sequence (seq2seq) modeling for Machine Translation (MT). The classic RNN-based approaches to MT were first out-performed by the convolutional seq2seq model, which was then out-performed by the more recent Transformer model. Each of these new approaches consists of a fundamental architecture accompanied by a set of modeling and training techniques that are in principle applicable to other seq2seq architectures. In this paper, we tease apart the new architectures and their accompanying techniques in two ways. First, we identify several key modeling and training techniques, and apply them to the RNN architecture, yielding a new RNMT+ model that outperforms all of the three fundamental architectures on the benchmark WMT'14 English→French and English→German tasks. Second, we analyze the properties of each fundamental seq2seq architecture and devise new hybrid architectures intended to combine their strengths. Our hybrid models obtain further improvements, outperforming the RNMT+ model on both benchmark datasets.

1 Introduction

In recent years, the emergence of seq2seq models (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014) has revolutionized the field of MT by replacing traditional phrase-based approaches with neural machine translation (NMT) systems based on the encoder-decoder paradigm. In the first architectures that surpassed the quality of phrase-based MT, both the encoder and decoder were implemented as Recurrent Neural Networks (RNNs), interacting via a soft-attention mechanism (Bahdanau et al., 2015). The RNN-based NMT approach, or RNMT, was quickly established as the de-facto standard for NMT, and gained rapid adoption into large-scale systems in industry, e.g. Baidu (Zhou et al., 2016), Google (Wu et al., 2016), and Systran (Crego et al., 2016).

Following RNMT, convolutional neural network based approaches (LeCun and Bengio, 1998) to NMT have recently drawn research attention due to their ability to fully parallelize training to take advantage of modern fast computing devices. such as GPUs and Tensor Processing Units (TPUs) (Jouppi et al., 2017). Well known examples are ByteNet (Kalchbrenner et al., 2016) and ConvS2S (Gehring et al., 2017). The ConvS2S model was shown to outperform the original RNMT architecture in terms of quality, while also providing greater training speed.

Most recently, the Transformer model (Vaswani et al., 2017), which is based solely on a self-attention mechanism (Parikh et al., 2016) and feed-forward connections, has further advanced the field of NMT, both in terms of translation quality and speed of convergence.

In many instances, new architectures are accompanied by a novel set of techniques for performing training and inference that have been carefully optimized to work in concert. This 'bag of tricks' can be crucial to the performance of a proposed architecture, yet it is typically under-documented and left for the enterprising researcher to discover in publicly released code (if any) or through anecdotal evidence. This is not simply a problem for reproducibility; it obscures the central scientific question of how much of the observed gains come from the new architecture

[*] Equal contribution.

and how much can be attributed to the associated training and inference techniques. In some cases, these new techniques may be broadly applicable to other architectures and thus constitute a major, though implicit, contribution of an architecture paper. Clearly, they need to be considered in order to ensure a fair comparison across different model architectures.

In this paper, we therefore take a step back and look at which techniques and methods contribute significantly to the success of recent architectures, namely ConvS2S and Transformer, and explore applying these methods to other architectures, including RNMT models. In doing so, we come up with an enhanced version of RNMT, referred to as RNMT+, that significantly outperforms all individual architectures in our setup. We further introduce new architectures built with different components borrowed from RNMT+, ConvS2S and Transformer. In order to ensure a fair setting for comparison, all architectures were implemented in the same framework, use the same pre-processed data and apply no further post-processing as this may confound bare model performance.

Our contributions are three-fold:

1. In ablation studies, we quantify the effect of several modeling improvements (including multi-head attention and layer normalization) as well as optimization techniques (such as synchronous replica training and label-smoothing), which are used in recent architectures. We demonstrate that these techniques are applicable across different model architectures.

2. Combining these improvements with the RNMT model, we propose the new RNMT+ model, which significantly outperforms all fundamental architectures on the widely-used WMT'14 En$\rightarrow$Fr and En$\rightarrow$De benchmark datasets. We provide a detailed model analysis and comparison of RNMT+, ConvS2S and Transformer in terms of model quality, model size, and training and inference speed.

3. Inspired by our understanding of the relative strengths and weaknesses of individual model architectures, we propose new model architectures that combine components from the RNMT+ and the Transformer model, and achieve better results than both individual architectures.

We quickly note two prior works that provided empirical solutions to the difficulty of training NMT architectures (specifically RNMT). In (Britz et al., 2017) the authors systematically explore which elements of NMT architectures have a significant impact on translation quality. In (Denkowski and Neubig, 2017) the authors recommend three specific techniques for strengthening NMT systems and empirically demonstrated how incorporating those techniques improves the reliability of the experimental results.

2 Background

In this section, we briefly discuss the commmonly used NMT architectures.

2.1 RNN-based NMT Models - RNMT

RNMT models are composed of an encoder RNN and a decoder RNN, coupled with an attention network. The encoder summarizes the input sequence into a set of vectors while the decoder conditions on the encoded input sequence through an attention mechanism, and generates the output sequence one token at a time.

The most successful RNMT models consist of stacked RNN encoders with one or more bidirectional RNNs, and stacked decoders with unidirectional RNNs. Both encoder and decoder RNNs consist of either LSTM (Hochreiter and Schmidhuber, 1997) or GRU units (Cho et al., 2014), and make extensive use of residual (He et al., 2015) or highway (Srivastava et al., 2015) connections.

In Google-NMT (GNMT) (Wu et al., 2016), the best performing RNMT model on the datasets we consider, the encoder network consists of one bi-directional LSTM layer, followed by 7 uni-directional LSTM layers. The decoder is equipped with a single attention network and 8 uni-directional LSTM layers. Both the encoder and the decoder use residual skip connections between consecutive layers.

In this paper, we adopt GNMT as the starting point for our proposed RNMT+ architecture, following the public NMT codebase[1].

2.2 Convolutional NMT Models - ConvS2S

In the most successful convolutional sequence-to-sequence model (Gehring et al., 2017), both the encoder and decoder are constructed by stacking multiple convolutional layers, where each layer

[1]https://github.com/tensorflow/nmt

contains 1-dimensional convolutions followed by a gated linear units (GLU) (Dauphin et al., 2016). Each decoder layer computes a separate dot-product attention by using the current decoder layer output and the final encoder layer outputs. Positional embeddings are used to provide explicit positional information to the model. Following the practice in (Gehring et al., 2017), we scale the gradients of the encoder layers to stabilize training. We also use residual connections across each convolutional layer and apply weight normalization (Salimans and Kingma, 2016) to speed up convergence. We follow the public ConvS2S codebase[2] in our experiments.

2.3 Conditional Transformation-based NMT Models - Transformer

The Transformer model (Vaswani et al., 2017) is motivated by two major design choices that aim to address deficiencies in the former two model families: (1) Unlike RNMT, but similar to the ConvS2S, the Transformer model avoids any sequential dependencies in both the encoder and decoder networks to maximally parallelize training. (2) To address the limited context problem (limited receptive field) present in ConvS2S, the Transformer model makes pervasive use of self-attention networks (Parikh et al., 2016) so that each position in the current layer has access to information from all other positions in the previous layer.

The Transformer model still follows the encoder-decoder paradigm. Encoder transformer layers are built with two sub-modules: (1) a self-attention network and (2) a feed-forward network. Decoder transformer layers have an additional cross-attention layer sandwiched between the self-attention and feed-forward layers to attend to the encoder outputs.

There are two details which we found very important to the model's performance: (1) Each sub-layer in the transformer (i.e. self-attention, cross-attention, and the feed-forward sub-layer) follows a strict computation sequence: *normalize* $\rightarrow$ *transform* $\rightarrow$ *dropout* $\rightarrow$ *residual-add*. (2) In addition to per-layer normalization, the final encoder output is again normalized to prevent a blow up after consecutive residual additions.

In this paper, we follow the latest version of the Transformer model in the public Tensor2Tensor[3] codebase.

2.4 A Theory-Based Characterization of NMT Architectures

From a theoretical point of view, RNNs belong to the most expressive members of the neural network family (Siegelmann and Sontag, 1995)[4]. Possessing an infinite Markovian structure (and thus an infinite receptive fields) equips them to model sequential data (Elman, 1990), especially natural language (Grefenstette et al., 2015) effectively. In practice, RNNs are notoriously hard to train (Bengio et al., 1994), confirming the well known dilemma of trainability versus expressivity.

Convolutional layers are adept at capturing local context and local correlations by design. A fixed and narrow receptive field for each convolutional layer limits their capacity when the architecture is shallow. In practice, this weakness is mitigated by stacking more convolutional layers (e.g. 15 layers as in the ConvS2S model), which makes the model harder to train and demands meticulous initialization schemes and carefully designed regularization techniques.

The transformer network is capable of approximating arbitrary squashing functions (Hornik et al., 1989), and can be considered a strong feature extractor with extended receptive fields capable of linking salient features from the entire sequence. On the other hand, lacking a memory component (as present in the RNN models) prevents the network from modeling a state space, reducing its theoretical strength as a sequence model, thus it requires additional positional information (e.g. sinusoidal positional encodings).

Above theoretical characterizations will drive our explorations in the following sections.

3 Experiment Setup

We train our models on the standard WMT'14 En$\rightarrow$Fr and En$\rightarrow$De datasets that comprise 36.3M and 4.5M sentence pairs, respectively. Each sentence was encoded into a sequence of sub-word units obtained by first tokenizing the sentence with the Moses tokenizer, then splitting tokens into sub-word units (also known as "wordpieces") using the approach described in (Schuster and Nakajima, 2012).

[2]https://github.com/facebookresearch/fairseq-py

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

[4]Assuming that data complexity is satisfied.

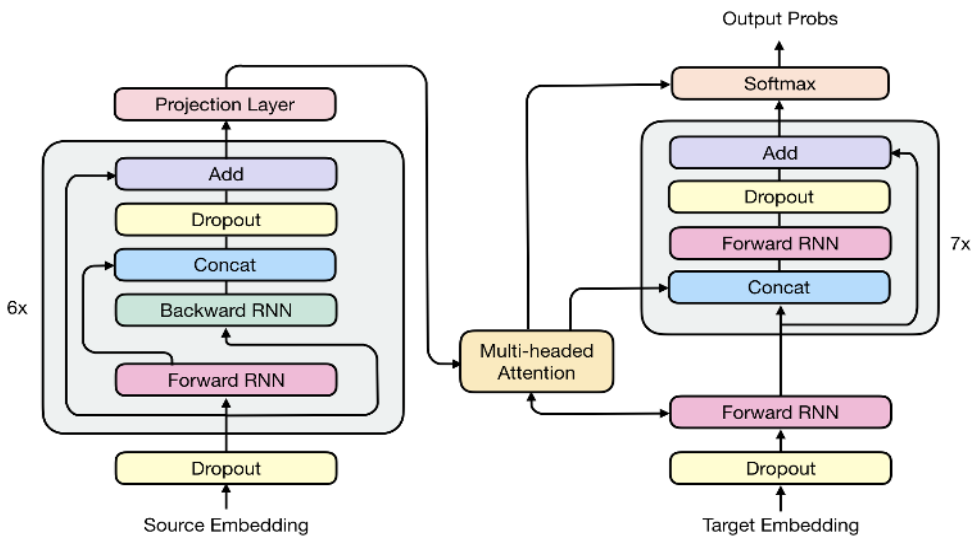


Figure 1: Model architecture of RNMT+. On the left side, the encoder network has 6 bidirectional LSTM layers. At the end of each bidirectional layer, the outputs of the forward layer and the backward layer are concatenated. On the right side, the decoder network has 8 unidirectional LSTM layers, with the first layer used for obtaining the attention context vector through multi-head additive attention. The attention context vector is then fed directly into the rest of the decoder layers as well as the softmax layer.

We use a shared vocabulary of 32K sub-word units for each source-target language pair. No further manual or rule-based post processing of the output was performed beyond combining the sub-word units to generate the targets. We report all our results on newstest 2014, which serves as the test set. A combination of newstest 2012 and newstest 2013 is used for validation.

To evaluate the models, we compute the BLEU metric on tokenized, true-case output.[5] For each training run, we evaluate the model every 30 minutes on the dev set. Once the model converges, we determine the best window based on the average dev-set BLEU score over 21 consecutive evaluations. We report the mean test score and standard deviation over the selected window. This allows us to compare model architectures based on their mean performance after convergence rather than individual checkpoint evaluations, as the latter can be quite noisy for some models.

To enable a fair comparison of architectures, we use the same pre-processing and evaluation methodology for all our experiments. We refrain from using checkpoint averaging (exponential moving averages of parameters) (Junczys-Dowmunt et al., 2016) or checkpoint ensembles (Jean et al., 2015; Chen et al., 2017) to focus on evaluating the performance of individual models.

4 RNMT+

4.1 Model Architecture of RNMT+

The newly proposed RNMT+ model architecture is shown in Figure 1. Here we highlight the key architectural choices that are different between the RNMT+ model and the GNMT model. There are 6 bidirectional LSTM layers in the encoder instead of 1 bidirectional LSTM layer followed by 7 unidirectional layers as in GNMT. For each bidirectional layer, the outputs of the forward layer and the backward layer are concatenated before being fed into the next layer. The decoder network consists of 8 unidirectional LSTM layers similar to the GNMT model. Residual connections are added to the third layer and above for both the encoder and decoder. Inspired by the Transformer model, per-gate layer normalization (Ba et al., 2016) is applied within each LSTM cell. Our empirical results show that layer normalization greatly stabilizes training. No non-linearity is applied to the LSTM output. A projection layer is added to the encoder final output.[6] Multi-head additive attention is used instead of the single-head attention in the GNMT model. Similar to GNMT, we use the

[5]This procedure is used in the literature to which we compare (Gehring et al., 2017; Wu et al., 2016).

[6]Additional projection aims to reduce the dimensionality of the encoder output representations to match the decoder stack dimension.

bottom decoder layer and the final encoder layer output after projection for obtaining the recurrent attention context. In addition to feeding the attention context to all decoder LSTM layers, we also feed it to the softmax. This is important for both the quality of the models with multi-head attention and the stability of the training process.

Since the encoder network in RNMT+ consists solely of bi-directional LSTM layers, model parallelism is not used during training. We compensate for the resulting longer per-step time with increased data parallelism (more model replicas), so that the overall time to reach convergence of the RNMT+ model is still comparable to that of GNMT.

We apply the following regularization techniques during training.

- **Dropout:** We apply dropout to both embedding layers and each LSTM layer output before it is added to the next layer's input. Attention dropout is also applied.

- **Label Smoothing:** We use uniform label smoothing with an uncertainty=0.1 (Szegedy et al., 2015). Label smoothing was shown to have a positive impact on both Transformer and RNMT+ models, especially in the case of RNMT+ with multi-head attention. Similar to the observations in (Chorowski and Jaitly, 2016), we found it beneficial to use a larger beam size (e.g. 16, 20, etc.) during decoding when models are trained with label smoothing.

- **Weight Decay:** For the WMT'14 En→De task, we apply L2 regularization to the weights with $\lambda = 10^{-5}$. Weight decay is only applied to the En→De task as the corpus is smaller and thus more regularization is required.

We use the Adam optimizer (Kingma and Ba, 2014) with $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-6}$ and vary the learning rate according to this schedule:

$$lr = 10^{-4} \cdot \min\left(1 + \frac{t \cdot (n-1)}{np}, n, n \cdot (2n)^{\frac{s-nt}{e-s}}\right) \tag{1}$$

Here, t is the current step, n is the number of concurrent model replicas used in training, p is the number of warmup steps, s is the start step of the exponential decay, and e is the end step of the decay. Specifically, we first increase the learning rate linearly during the number of warmup steps, keep it a constant until the decay start step s, then exponentially decay until the decay end step e, and keep it at $5 \cdot 10^{-5}$ after the decay ends. This learning rate schedule is motivated by a similar schedule that was successfully applied in training the Resnet-50 model with a very large batch size (Goyal et al., 2017).

In contrast to the asynchronous training used for GNMT (Dean et al., 2012), we train RNMT+ models with synchronous training (Chen et al., 2016). Our empirical results suggest that when hyper-parameters are tuned properly, synchronous training often leads to improved convergence speed and superior model quality.

To further stabilize training, we also use adaptive gradient clipping. We discard a training step completely if an anomaly in the gradient norm value is detected, which is usually an indication of an imminent gradient explosion. More specifically, we keep track of a moving average and a moving standard deviation of the $\log$ of the gradient norm values, and we abort a step if the norm of the gradient exceeds four standard deviations of the moving average.

4.2 Model Analysis and Comparison

In this section, we compare the results of RNMT+ with ConvS2S and Transformer.

All models were trained with synchronous training. RNMT+ and ConvS2S were trained with 32 NVIDIA P100 GPUs while the Transformer Base and Big models were trained using 16 GPUs.

For RNMT+, we use sentence-level cross-entropy loss. Each training batch contained 4096 sentence pairs (4096 source sequences and 4096 target sequences). For ConvS2S and Transformer models, we use token-level cross-entropy loss. Each training batch contained 65536 source tokens and 65536 target tokens. For the GNMT baselines on both tasks, we cite the largest BLEU score reported in (Wu et al., 2016) without reinforcement learning.

Table 1 shows our results on the WMT'14 En→Fr task. Both the Transformer Big model and RNMT+ outperform GNMT and ConvS2S by about 2 BLEU points. RNMT+ is slightly better than the Transformer Big model in terms of its mean BLEU score. RNMT+ also yields a much lower standard deviation, and hence we observed much less fluctuation in the training curve. It takes approximately 3 days for the Transformer

Base model to converge, while both RNMT+ and the Transformer Big model require about 5 days to converge. Although the batching schemes are quite different between the Transformer Big and the RNMT+ model, they have processed about the same amount of training samples upon convergence.

Model	Test BLEU	Epochs	Training Time
GNMT	38.95	-	-
ConvS2S [7]	39.49 ± 0.11	62.2	438h
Trans. Base	39.43 ± 0.17	20.7	90h
Trans. Big [8]	40.73 ± 0.19	8.3	120h
RNMT+	41.00 ± 0.05	8.5	120h

Table 1: Results on WMT14 En→Fr. The numbers before and after '±' are the mean and standard deviation of test BLEU score over an evaluation window.

Table 2 shows our results on the WMT'14 En→De task. The Transformer Base model improves over GNMT and ConvS2S by more than 2 BLEU points while the Big model improves by over 3 BLEU points. RNMT+ further outperforms the Transformer Big model and establishes a new state of the art with an averaged value of 28.49. In this case, RNMT+ converged slightly faster than the Transformer Big model and maintained much more stable performance after convergence with a very small standard deviation, which is similar to what we observed on the En-Fr task.

Table 3 summarizes training performance and model statistics. The Transformer Base model is the fastest model in terms of training speed. RNMT+ is slower to train than the Transformer

[7]Since the ConvS2S model convergence is very slow we did not explore further tuning on En→Fr, and validated our implementation on En→De.

[8]The BLEU scores for Transformer model are slightly lower than those reported in (Vaswani et al., 2017) due to four differences:

1) We report the mean test BLEU score using the strategy described in section 3.

2) We did not perform checkpoint averaging since it would be inconsistent with our evaluation for other models.

3) We avoided any manual post-processing, like unicode normalization using Moses replace-unicode-punctuation.perl or output tokenization using Moses tokenizer.perl, to rule out its effect on the evaluation. We observed a significant BLEU increase (about 0.6) on applying these post processing techniques.

4) In (Vaswani et al., 2017), reported BLEU scores are calculated using mteval-v13a.pl from Moses, which re-tokenizes its input.

Model	Test BLEU	Epochs	Training Time
GNMT	24.67	-	-
ConvS2S	25.01 ±0.17	38	20h
Trans. Base	27.26 ± 0.15	38	17h
Trans. Big	27.94 ± 0.18	26.9	48h
RNMT+	28.49 ± 0.05	24.6	40h

Table 2: Results on WMT14 En→De.

Big model on a per-GPU basis. However, since the RNMT+ model is quite stable, we were able to offset the lower per-GPU throughput with higher concurrency by increasing the number of model replicas, and hence the overall time to convergence was not slowed down much. We also computed the number of floating point operations (FLOPs) in the model's forward path as well as the number of total parameters for all architectures (cf. Table 3). RNMT+ requires fewer FLOPs than the Transformer Big model, even though both models have a comparable number of parameters.

Model	Examples/s	FLOPs	Params
ConvS2S	80	15.7B	263.4M
Trans. Base	160	6.2B	93.3M
Trans. Big	50	31.2B	375.4M
RNMT+	30	28.1B	378.9M

Table 3: Performance comparison. Examples/s are normalized by the number of GPUs used in the training job. FLOPs are computed assuming that source and target sequence length are both 50.

5 Ablation Experiments

In this section, we evaluate the importance of four main techniques for both the RNMT+ and the Transformer Big models. We believe that these techniques are universally applicable across different model architectures, and should always be employed by NMT practitioners for best performance.

We take our best RNMT+ and Transformer Big models and remove each one of these techniques independently. By doing this we hope to learn two things about each technique: (1) How much does it affect the model performance? (2) How useful is it for stable training of other techniques and hence the final model?

From Table 4 we draw the following conclusions about the four techniques:

Model	RNMT+	Trans. Big
Baseline	41.00	40.73
- Label Smoothing	40.33	40.49
- Multi-head Attention	40.44	39.83
- Layer Norm.	*	*
- Sync. Training	39.68	*

Table 4: Ablation results of RNMT+ and the Transformer Big model on WMT'14 En $\rightarrow$ Fr. We report average BLEU scores on the test set. An asterisk '*' indicates an unstable training run (training halts due to non-finite elements).

- **Label Smoothing** We observed that label smoothing improves both models, leading to an average increase of 0.7 BLEU for RNMT+ and 0.2 BLEU for Transformer Big models.
- **Multi-head Attention** Multi-head attention contributes significantly to the quality of both models, resulting in an average increase of 0.6 BLEU for RNMT+ and 0.9 BLEU for Transformer Big models.
- **Layer Normalization** Layer normalization is most critical to stabilize the training process of either model, especially when multi-head attention is used. Removing layer normalization results in unstable training runs for both models. Since by design, we remove one technique at a time in our ablation experiments, we were unable to quantify how much layer normalization helped in either case. To be able to successfully train a model without layer normalization, we would have to adjust other parts of the model and retune its hyper-parameters.
- **Synchronous training** Removing synchronous training has different effects on RNMT+ and Transformer. For RNMT+, it results in a significant quality drop, while for the Transformer Big model, it causes the model to become unstable. We also notice that synchronous training is only successful when coupled with a tailored learning rate schedule that has a warmup stage at the beginning (cf. Eq. 1 for RNMT+ and Eq. 2 for Transformer). For RNMT+, removing this warmup stage during synchronous training causes the model to become unstable.

6 Hybrid NMT Models

In this section, we explore hybrid architectures that shed some light on the salient behavior of each model family. These hybrid models outperform the individual architectures on both benchmark datasets and provide a better understanding of the capabilities and limitations of each model family.

6.1 Assessing Individual Encoders and Decoders

In an encoder-decoder architecture, a natural assumption is that the role of an encoder is to build feature representations that can best encode the meaning of the source sequence, while a decoder should be able to process and interpret the representations from the encoder and, at the same time, track the current target history. Decoding is inherently auto-regressive, and keeping track of the state information should therefore be intuitively beneficial for conditional generation.

We set out to study which family of encoders is more suitable to extract rich representations from a given input sequence, and which family of decoders can make the best of such rich representations. We start by combining the encoder and decoder from different model families. Since it takes a significant amount of time for a ConvS2S model to converge, and because the final translation quality was not on par with the other models, we focus on two types of hybrids only: Transformer encoder with RNMT+ decoder and RNMT+ encoder with Transformer decoder.

Encoder	Decoder	En$\rightarrow$Fr Test BLEU
Trans. Big	Trans. Big	40.73 $\pm$ 0.19
RNMT+	RNMT+	41.00 $\pm$ 0.05
Trans. Big	RNMT+	**41.12 $\pm$ 0.16**
RNMT+	Trans. Big	39.92 $\pm$ 0.21

Table 5: Results for encoder-decoder hybrids.

From Table 5, it is clear that the Transformer encoder is better at encoding or feature extraction than the RNMT+ encoder, whereas RNMT+ is better at decoding or conditional language modeling, confirming our intuition that a stateful decoder is beneficial for conditional language generation.

6.2 Assessing Encoder Combinations

Next, we explore how the features extracted by an encoder can be further enhanced by incorporating additional information. Specifically, we investigate the combination of transformer layers with RNMT+ layers in the same encoder block to build even richer feature representations. We exclusively use RNMT+ decoders in the following architectures since stateful decoders show better performance according to Table 5.

We study two mixing schemes in the encoder (see Fig. 2):

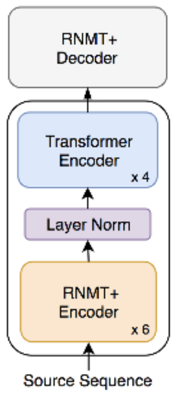
(1) *Cascaded Encoder*: The cascaded encoder aims at combining the representational power of RNNs and self-attention. The idea is to enrich a set of stateful representations by cascading a feature extractor with a focus on vertical mapping, similar to (Pascanu et al., 2013; Devlin, 2017). Our best performing cascaded encoder involves fine tuning transformer layers stacked on top of a pre-trained frozen RNMT+ encoder. Using a pre-trained encoder avoids optimization difficulties while significantly enhancing encoder capacity. As shown in Table 6, the cascaded encoder improves over the Transformer encoder by more than 0.5 BLEU points on the WMT'14 En→Fr task. This suggests that the Transformer encoder is able to extract richer representations if the input is augmented with sequential context.

(2) *Multi-Column Encoder*: As illustrated in Fig. 2b, a multi-column encoder merges the outputs of several independent encoders into a single combined representation. Unlike a cascaded encoder, the multi-column encoder enables us to investigate whether an RNMT+ decoder can distinguish information received from two different channels and benefit from its combination. A crucial operation in a multi-column encoder is therefore how different sources of information are merged into a unified representation. Our best multi-column encoder performs a simple concatenation of individual column outputs.

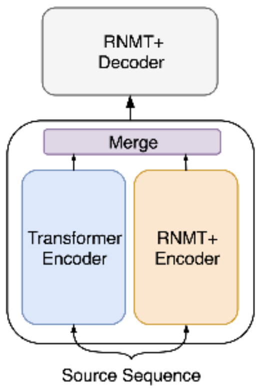
The model details and hyperparameters of the above two encoders are described in Appendix A.5 and A.6. As shown in Table 6, the multi-column encoder followed by an RNMT+ decoder achieves better results than the Transformer and the RNMT model on both WMT'14 benchmark tasks.

Model	En→Fr BLEU	En→De BLEU
Trans. Big	40.73 ± 0.19	27.94 ± 0.18
RNMT+	41.00 ± 0.05	28.49 ± 0.05
Cascaded	**41.67 ± 0.11**	28.62 ± 0.06
MultiCol	41.66 ± 0.11	**28.84 ± 0.06**

Table 6: Results for hybrids with cascaded encoder and multi-column encoder.



(a) Cascaded Encoder



(b) Multi-Column Encoder

Figure 2: Vertical and horizontal mixing of Transformer and RNMT+ components in an encoder.

7 Conclusion

In this work we explored the efficacy of several architectural and training techniques proposed in recent studies on seq2seq models for NMT. We demonstrated that many of these techniques are broadly applicable to multiple model architectures. Applying these new techniques to RNMT models yields RNMT+, an enhanced RNMT model that significantly outperforms the three fundamental architectures on WMT'14 En→Fr and En→De tasks. We further presented several hybrid models developed by combining encoders and decoders from the Transformer and RNMT+ models, and empirically demonstrated the superiority of the Transformer encoder and the RNMT+ decoder in comparison with their counterparts. We then enhanced the encoder architecture by horizontally and vertically mixing components borrowed from these architectures, leading to hybrid architectures that obtain further improvements over RNMT+.

We hope that our work will motivate NMT researchers to further investigate generally applicable training and optimization techniques, and that our exploration of hybrid architectures will open paths for new architecture search efforts for NMT.

Our focus on a standard single-language-pair translation task leaves important open questions to be answered: How do our new architectures compare in multilingual settings, i.e., modeling an *interlingua*? Which architecture is more efficient and powerful in processing finer grained inputs and outputs, e.g., characters or bytes? How transferable are the representations learned by the different architectures to other tasks? And what are the characteristic errors that each architecture makes, e.g., linguistic plausibility?

Acknowledgments

We would like to thank the entire Google Brain Team and Google Translate Team for their foundational contributions to this project. We would also like to thank Noam Shazeer, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, and the entire Tensor2Tensor development team for their useful inputs and discussions.

References

Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *CoRR* abs/1607.06450. http://arxiv.org/abs/1607.06450.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*. http://arxiv.org/abs/1409.0473.

Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.* 5(2):157–166. https://doi.org/10.1109/72.279181.

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. https://www.aclweb.org/anthology/D17-1151.

Hugh Chen, Scott Lundberg, and Su-In Lee. 2017. Checkpoint ensembles: Ensemble methods from a single training process. *CoRR* abs/1710.03282. http://arxiv.org/abs/1710.03282.

Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Józefowicz. 2016. Revisiting distributed synchronous SGD. *CoRR* abs/1604.00981. http://arxiv.org/abs/1604.00981.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*. http://arxiv.org/abs/1406.1078.

Jan Chorowski and Navdeep Jaitly. 2016. Towards better decoding and language model integration in sequence to sequence models. *CoRR* abs/1612.02695. http://arxiv.org/abs/1612.02695.

Josep Maria Crego, Jungi Kim, Guillaume Klein, Anabel Rebollo, Kathy Yang, Jean Senellart, Egor Akhanov, Patrice Brunelle, Aurelien Coquard, Yongchao Deng, Satoshi Enoue, Chiyo Geiss, Joshua Johanson, Ardas Khalsa, Raoum Khiari, Byeongil Ko, Catherine Kobus, Jean Lorieux, Leidiana Martins, Dang-Chuan Nguyen, Alexandra Priori, Thomas Riccardi, Natalia Segal, Christophe Servan, Cyril Tiquet, Bo Wang, Jin Yang, Dakun Zhang, Jing Zhou, and Peter Zoldan. 2016. Systran's pure neural machine translation systems. *CoRR* abs/1610.05540. http://arxiv.org/abs/1610.05540.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2016. Language modeling with gated convolutional networks. *CoRR* abs/1612.08083. http://arxiv.org/abs/1612.08083.

Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, MarcAurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large scale distributed deep networks. In *NIPS*.

Michael Denkowski and Graham Neubig. 2017. Stronger baselines for trustable results in neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*. Association for Computational Linguistics, Vancouver, pages 18–27. http://www.aclweb.org/anthology/W17-3203.

Jacob Devlin. 2017. Sharp models on dull hardware: Fast and accurate neural machine translation decoding on the cpu. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 2820–2825. http://aclweb.org/anthology/D17-1300.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science* 14(2):179 – 211. https://doi.org/https://doi.org/10.1016/0364-0213(90)90002-E.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning. *CoRR* abs/1705.03122. http://arxiv.org/abs/1705.03122.

Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR* abs/1706.02677. http://arxiv.org/abs/1706.02677.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to transduce with unbounded memory. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. MIT Press, Cambridge, MA, USA, NIPS'15, pages 1828–1836. http://dl.acm.org/citation.cfm?id=2969442.2969444.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *CoRR* abs/1512.03385. http://arxiv.org/abs/1512.03385.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2(5):359 – 366. https://doi.org/https://doi.org/10.1016/0893-6080(89)90020-8.

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*.

Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, and et al. 2017. In-datacenter performance analysis of a tensor processing unit. *CoRR* abs/1704.04760. http://arxiv.org/abs/1704.04760.

Marcin Junczys-Dowmunt, Tomasz Dwojak, and Rico Sennrich. 2016. The amu-uedin submission to the wmt16 news translation task: Attention-based nmt models as feature functions in phrase-based smt. *arXiv preprint arXiv:1605.04809* .

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *Conference on Empirical Methods in Natural Language Processing*.

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. http://arxiv.org/abs/1610.10099.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980. http://arxiv.org/abs/1412.6980.

Yann LeCun and Yoshua Bengio. 1998. The handbook of brain theory and neural networks. MIT Press, Cambridge, MA, USA, chapter Convolutional Networks for Images, Speech, and Time Series, pages 255–258. http://dl.acm.org/citation.cfm?id=303568.303704.

Ankur P. Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *EMNLP*.

Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. 2013. How to construct deep recurrent neural networks. *CoRR* abs/1312.6026. http://arxiv.org/abs/1312.6026.

Tim Salimans and Diederik P. Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR* abs/1602.07868. http://arxiv.org/abs/1602.07868.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and Korean voice search. *2012 IEEE International Conference on Acoustics, Speech and Signal Processing* .

H.T. Siegelmann and E.D. Sontag. 1995. On the computational power of neural nets. *Journal of Computer and System Sciences* 50(1):132 – 150. https://doi.org/https://doi.org/10.1006/jcss.1995.1013.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks. *CoRR* abs/1505.00387. http://arxiv.org/abs/1505.00387.

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. 2015. Rethinking the inception architecture for computer vision. *CoRR* abs/1512.00567. http://arxiv.org/abs/1512.00567.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR* abs/1706.03762. http://arxiv.org/abs/1706.03762.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR* abs/1609.08144. http://arxiv.org/abs/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. *CoRR* abs/1606.04199. http://arxiv.org/abs/1606.04199.

A Supplemental Material

A.1 ConvS2S

For the WMT'14 En→De task, both the encoder and decoder have 15 layers, with 512 hidden units in the first ten layers, 768 units in the subsequent three layers and 2048 units in the final two layers. The first 13 layers use kernel width 3 and the final two layers use kernel width 1. For the WMT'14 En→Fr task, both the encoder and decoder have 14 layers, with 512 hidden units in the first five layers, 768 units in the subsequent four layers, 1024 units in the next three layers, 2048 units and 4096 units in the final two layers. The first 12 layers use kernel width 3 and the final two layers use kernel width 1. We train the ConvS2S models with synchronous training using 32 GPUs.

A.2 Transformer

Both the encoder and the decoder have 6 Transformer layers. Transformer base model has model dimension 512, hidden dimension 2048 and 8 attention heads. The Transformer Big model uses model dimension 1024, hidden dimension 8192 and 16 attention heads. We group the dropout in Transformer models into four types: *input dropout* - dropout applied to the sum of token embeddings and position encodings, *residual dropout* - dropout applied to the output of each sublayer before added to the sublayer input, *relu dropout* - dropout applied to the inner layer output after ReLU activation in each feed-forward sub-layer, *attention dropout* - dropout applied to attention weight in each attention sub-layer. All Transformer models use the following learning rate schedule:

$$lr = \frac{r_0}{\sqrt{d_{model}}} \cdot \min\left(\frac{t+1}{p\sqrt{p}}, \frac{1}{\sqrt{(t+1)}}\right) \qquad (2)$$

where t is the current step, p is the number of warmup steps, d_{model} is the model dimension and r_0 is a constant to adjust the magnitude of the learning rate.

On WMT'14 En→De, the Transformer Base model employs all four types of dropout with $dropout\_probs = 0.1$. We use $r_0 = 2.0$ and $p = 8000$ in the learning rate schedule. For the Transformer Big model, only residual dropout and input dropout are applied, both with $dropout\_probs = 0.3$. $r_0 = 3.0$ and $p = 40000$ are used in the learning rate schedule.

On WMT'14 En→Fr, the Base model applies only residual dropout and input dropout, each with $dropout\_probs = 0.1$. The learning rate schedule uses $r_0 = 1.0$ and $p = 4000$. For the big model, we apply all four types of dropout, each with $dropout\_probs = 0.1$. The learning rate schedule uses $r_0 = 3.0$ and $p = 40000$.

We train both Transformer base model and big model with synchronous training using 16 GPUs.

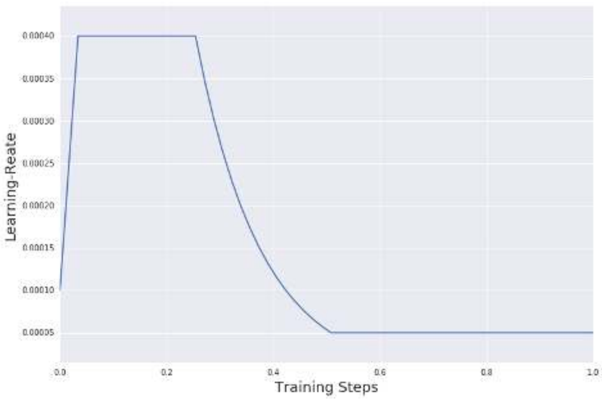


Figure 3: RNMT+ learning-rate schedule.

A.3 RNMT+

RNMT+ has 1024 LSTM nodes in all encoder and decoder layers. The input embedding dimension is 1024 as well. The encoder final projection layer projects the last bidirectional layer output from dimension 2048 to 1024. We use 4 attention heads in the multi-head additive attention. Label smoothing is applied with an $uncertainty = 0.1$. Figure 3 illustrates our learning rate schedule defined in Eq. 1.

On WMT'14 En→De, we use $p = 500$, $s = 600000$, $e = 1200000$ for the learning rate schedule and apply all dropout types with $dropout\_probs = 0.3$. We apply L2 regularization to the weights with $\lambda = 10^{-5}$. On WMT'14 En→Fr, we use $p = 500$, $s = 1200000$, $e = 3600000$, $dropout\_probs = 0.2$. No weight decay is applied.

RNMT+ models are trained with synchronous training using 32 GPUs.

A.4 Encoder-Decoder Hybrids

For both encoder-decoder hybrids, i.e., Transformer Big encoder with RNMT+ decoder and RNMT+ encoder with Transformer Big decoder, we use the exactly same model hyperparameters as in the Transformer Big and RNMT+ models described in above sections.

We use Transformer learning rate schedule (Eq.

2) for both hybrids. For the WMT'14 En→Fr task, we use $r_0 = 4.0$ and $p = 50000$ for the hybrid with Transformer encoder and RNMT+ decoder, and use $r_0 = 3.0$ and $p = 40000$ for the hybrid with RNMT+ encoder and Transformer decoder. Both hybrid models are trained with synchronous training using 32 GPUs.

A.5 Cascaded Encoder Hybrid

In this hybrid we stack a transformer encoder on top of the RNMT+ encoder. In our experiments we used a pre-trained RNMT+ encoder, including the projection layer, exactly as described in section 4. The outputs of the RNMT+ encoder are layer normalized and fed into a transformer encoder. This structure is illustrated in Figure 2a. The transformer encoder is identical to the one described in subsection 2.3 except for the different number of layers. Our best setup uses 4 Transformer layers stacked on top of a pre-trained RNMT+ encoder with 6 layers. To speed up convergence, we froze gradient updates in the pre-trained RNMT+ encoder. This enables us to increase the encoder capacity significantly, while avoiding optimization issues encountered in non-frozen variants of the hybrid. As an additional benefit, this enables us to train the model on P100s without the need for model parallelism.

Note that this specific layout allows us to drop hand-crafted sinusoidal positional embeddings (since position information is already captured by the underlying RNNs).

We use the Transformer learning rate schedule (Eq. 2) for this hybrid with $r_0 = 2.0$ and $p = 16000$ and train with synchronous training using 32 GPUs. We apply the same dropouts used for the transformer model to the transformer layers in the encoder, and apply L2 weight decay with $\lambda = 10^{-5}$ to the decoder layers.

A.6 Multi-Column Encoder Hybrid

We use a simple concatenation as the merger-operator without fine-tuning any other model hyperparameters. After concatenation, the combined representation is projected down to the decoder dimension with a layer-normalized affine transformation. Although in this paper we only use two columns, there is no practical restriction on the total number of columns that this hybrid can combine. By combining multiple encoder representations, the network may capture different factors of variations in the input sequence.

Similar to the Cascaded-RNMT+ hybrid, we use pre-trained encoders that are borrowed from an RNMT+ model (we used a pretrained RNMT+ encoder as the first column) and an Encoder-Decoder hybrid model with Transformer encoder and RNMT+ decoder (we used the pretrained Transformer encoder). Multi-column encoder with RNMT+ decoder is trained using 16 GPUs in a synchronous training setup. We stick to the simple concatenation operation as the merger-operator, and after concatenation, the combined representation is projected down the decoder dimension with a simple layer-normalized affine transformation. One additional note that we observed for the sake of stability and trainability, each column output should be first mapped to a space where the representation ranges are compatible, e.g., RNMT+ encoder output has not limitation on its range, but a Transformer Encoder output range is constrained by the final layer normalization applied to the entire Transformer encoder body. Therefore, we also applied layer normalization to the RNMT+ encoder outputs to match the ranges of individual encoders.

On WMT'14 En→De, we use $p = 50$, $s = 300000$, $e = 900000$ for the learning rate schedule and apply all dropout types with $dropout\_probs = 0.3$. We apply L2 regularization to the weights with $\lambda = 10^{-5}$. On WMT'14 En→Fr, we use Transformer learning rate schedule (Eq. 2) $r_0 = 1.0$ and $p = 10000$. No weight decay or dropout is applied.