Massive Exploration of Neural Machine Translation Architectures

Denny Britz[*][†], Anna Goldie[*], Minh-Thang Luong, Quoc Le
{dennybritz,agoldie,thangluong,qvl}@google.com
Google Brain

Abstract

Neural Machine Translation (NMT) has shown remarkable progress over the past few years with production systems now being deployed to end-users. One major drawback of current architectures is that they are expensive to train, typically requiring days to weeks of GPU time to converge. This makes exhaustive hyperparameter search, as is commonly done with other neural network architectures, prohibitively expensive. In this work, we present the first large-scale analysis of NMT architecture hyperparameters. We report empirical results and variance numbers for several hundred experimental runs, corresponding to over 250,000 GPU hours on the standard WMT English to German translation task. Our experiments lead to novel insights and practical advice for building and extending NMT architectures. As part of this contribution, we release an open-source NMT framework[1] that enables researchers to easily experiment with novel techniques and reproduce state of the art results.

1 Introduction

Neural Machine Translation (NMT) (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014) is an end-to-end approach to automated translation. NMT has shown impressive results (Jean et al., 2015; Luong et al., 2015b; Sennrich et al., 2016a; Wu et al., 2016) surpassing those of phrase-based systems while addressing shortcomings such as the need for hand-engineered features. The most popular approaches to NMT are based on an encoder-decoder architecture consisting of two recurrent neural networks (RNNs) and an attention mechanism that aligns target with source tokens (Bahdanau et al., 2015; Luong et al., 2015a).

One shortcoming of current NMT architectures is the amount of compute required to train them. Training on real-world datasets of several million examples typically requires dozens of GPUs and convergence time is on the order of days to weeks (Wu et al., 2016). While sweeping across large hyperparameter spaces is common in Computer Vision (Huang et al., 2016b), such exploration would be prohibitively expensive for NMT models, limiting researchers to well-established architectures and hyperparameter choices. Furthermore, there have been no large-scale studies of how architectural hyperparameters affect the performance of NMT systems. As a result, it remains unclear why these models perform as well as they do, as well as how we might improve them.

In this work, we present the first comprehensive analysis of architectural hyperparameters for Neural Machine Translation systems. Using a total of more than 250,000 GPU hours, we explore common variations of NMT architectures and provide insight into which architectural choices matter most. We report BLEU scores, perplexities, model sizes, and convergence time for all experiments, including variance numbers calculated across several runs of each experiment. In addition, we release to the public a new software framework that was used to run the experiments.

In summary, the main contributions of this work are as follows:

- We provide immediately applicable insights into the optimization of Neural Machine Translation models, as well as promising directions for future research. For example, we

---

found that deep encoders are more difficult to optimize than decoders, that dense residual connections yield better performance than regular residual connections, that LSTMs outperform GRUs, and that a well-tuned beam search is crucial to obtaining state of the art results. By presenting practical advice for choosing baseline architectures, we help researchers avoid wasting time on unpromising model variations.

- We also establish the extent to which metrics such as BLEU are influenced by random initialization and slight hyperparameter variation, helping researchers to distinguish statistically significant results from random noise.

- Finally, we release an open source package based on TensorFlow, specifically designed for implementing reproducible state of the art sequence-to-sequence models. All experiments were run using this framework and we hope to accelerate future research by releasing it to the public. We also release all configuration files and processing scripts needed to reproduce the experiments in this paper.

2 Background and Preliminaries

2.1 Neural Machine Translation

Our models are based on an encoder-decoder architecture with attention mechanism (Bahdanau et al., 2015; Luong et al., 2015a), as shown in figure 1. An encoder function f_{enc} takes as input a sequence of source tokens $\mathbf{x} = (x_1, ..., x_m)$ and produces a sequence of states $\mathbf{h} = (h_1, ..., h_m)$. In our base model, f_{enc} is a bi-directional RNN and the state h_i corresponds to the concatenation of the states produced by the backward and forward RNNs, $h_i = [\overrightarrow{h_i}; \overleftarrow{h_i}]$. The decoder f_{dec} is an RNN that predicts the probability of a target sequence $\mathbf{y} = (y_1, ..., y_k)$ based on $\mathbf{h}$. The probability of each target token $y_i \in 1, ...V$ is predicted based on the recurrent state in the decoder RNN s_i, the previous words, $y_{<i}$, and a context vector c_i. The context vector c_i is also called the attention vector and is calculated as a weighted average of the source states.

$$c_i = \sum_j a_{ij} h_j \qquad (1)$$

$$a_{ij} = \frac{\hat{a}_{ij}}{\sum_j \hat{a}_{ij}} \qquad (2)$$

$$\hat{a}_{ij} = att(s_i, h_j) \qquad (3)$$

Here, $att(s_i, h_j)$ is an attention function that calculates an unnormalized alignment score between the encoder state h_j and the decoder state s_i. In our base model, we use a function of the form $att(s_i, h_j) = \langle W_h h_j, W_s s_i \rangle$, where the matrices W are used to transform the source and target states into a representation of the same size.

The decoder outputs a distribution over a vocabulary of fixed-size V:

$$P(y_i | y_1, ..., y_{i-1}, \mathbf{x})$$
$$= \text{softmax}(W[s_i; c_i] + b)$$

The whole model is trained end-to-end by minimizing the negative log likelihood of the target words using stochastic gradient descent.

3 Experimental Setup

3.1 Datasets and Preprocessing

We run all experiments on the WMT'15 English→German task consisting of 4.5M sentence pairs, obtained by combining the Europarl v7, News Commentary v10, and Common Crawl corpora. We use newstest2013 as our validation set and newstest2014 and newstest2015 as our test sets. To test for generality, we also ran a small number of experiments on English→French translation, and we found that the performance was highly correlated with that of English→German but that it took much longer to train models on the larger English→French dataset. Given that translation from the morphologically richer German is also considered a more challenging task, we felt justified in using the English→German translation task for this hyperparameter sweep.

We tokenize and clean all datasets with the scripts in Moses[2] and learn shared subword units using Byte Pair Encoding (BPE) (Sennrich et al., 2016b) using 32,000 merge operations for a final vocabulary size of approximately 37k. We discovered that data preprocessing can have a large impact on final numbers, and since we wish to enable

---

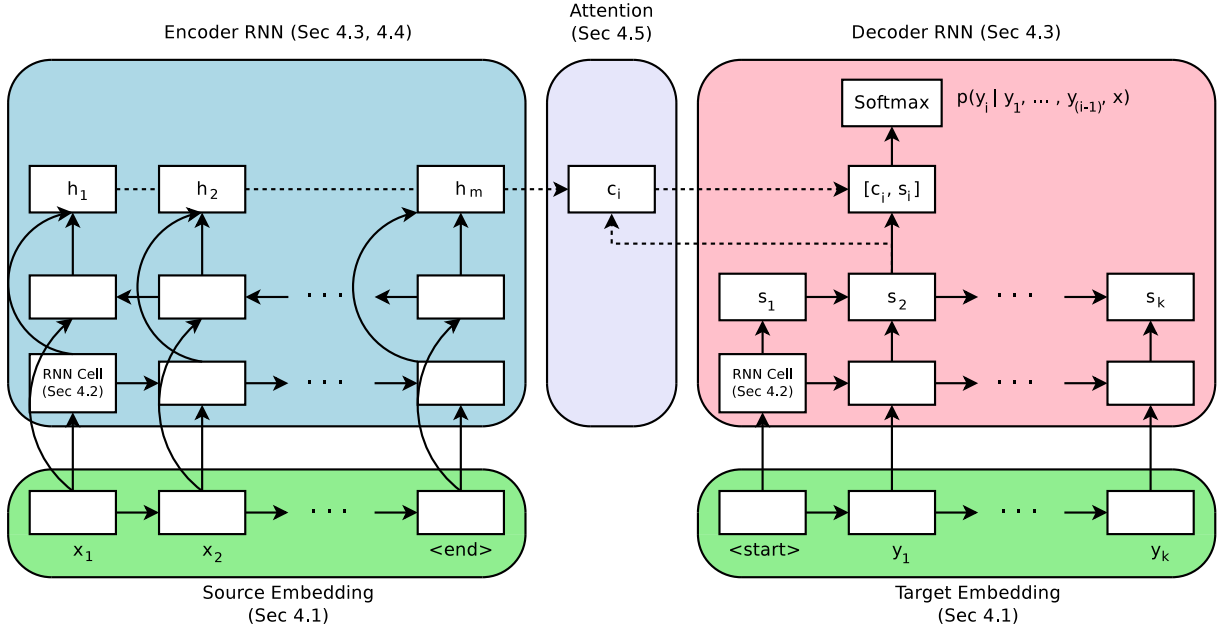[2]https://github.com/moses-smt/mosesdecoder/

Figure 1: Encoder-Decoder architecture with attention module. Section numbers reference experiments corresponding to the components.

reproducibility, we release our data preprocessing scripts together with the NMT framework to the public. For more details on data preprocessing parameters, we refer the reader to the code release.

3.2 Training Setup and Software

All of the following experiments are run using our own software framework based on TensorFlow (Abadi et al., 2016). We purposely built this framework to enable reproducible state-of-the-art implementations of Neural Machine Translation architectures. As part of our contribution, we are releasing the framework and all configuration files needed to reproduce our results. Training is performed on Nvidia Tesla K40m and Tesla K80 GPUs, distributed over 8 parallel workers and 6 parameter servers per experiment. We use a batch size of 128 and decode using beam search with a beam width of 10 and the length normalization penalty of 0.6 described in (Wu et al., 2016). BLEU scores are calculated on tokenized data using the *multi-bleu.perl* script in Moses[3]. Each experiment is run for a maximum of 2.5M steps and replicated 4 times with different initializations. We save model checkpoints every 30 minutes and choose the best checkpoint based on the validation set BLEU score. We report mean and standard de-

viation as well as highest scores (as per cross validation) for each experiment.

3.3 Baseline Model

Based on a review of previous literature, we chose a baseline model that we knew would perform reasonably well. Our goal was to keep the baseline model simple and standard, not to advance the start of the art. The model (described in 2.1) consists of a 2-layer bidirectional encoder (1 layer in each direction), and a 2 layer decoder with a multiplicative (Luong et al., 2015a) attention mechanism. We use 512-unit GRU (Cho et al., 2014) cells for both the encoder and decoder and apply Dropout of 0.2 at the input of each cell. We train using the Adam optimizer and a fixed learning rate of 0.0001 without decay. The embedding dimensionality is set to 512. A more detailed description of all model hyperparameters can be found in the supplementary material.

In each of the following experiments, the hyperparameters of the baseline model are held constant, except for the one hyperparameter being studied. We hope that this allows us to isolate the effect of various hyperparameter changes. We recognize that this procedure does not account for interactions between hyperparameters, and we perform additional experiments when we believe such interactions are likely to occur (e.g. skip connections and number of layers).

---

[3]https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl

4 Experiments and Discussion

For the sake of brevity, we only report mean BLEU, standard deviation, highest BLEU in parantheses, and model size in the following tables. Log perplexity, tokens/sec and convergence times can be found in the supplementary material tables.

4.1 Embedding Dimensionality

With a large vocabulary, the embedding layer can account for a large fraction of the model parameters. Historically, researchers have used 620-dimensional (Bahdanau et al., 2015) or 1024-dimensional (Luong et al., 2015a) embeddings. We expected larger embeddings to result in better BLEU scores, or at least lower perplexities, but we found that this wasn't always the case. While Table 1 shows that *2048-dimensional embeddings yielded the overall best result, they only did so by a small margin*. Even small 128-dimensional embeddings performed surprisingly well, while converging almost twice as quickly. We found that gradient updates to both small and large embeddings did not differ significantly and that the norm of gradient updates to the embedding matrix stayed approximately constant throughout training regardless of size. We also did not observe overfitting with large embeddings and training log perplexity was approximately equal across experiments, suggesting that the model does not make efficient use of the extra parameters and that there may be a need for better optimization techniques. Alternatively, it could be the case that models with large embeddings simply need much more than 2.5M steps to converge to the best solution.

Dim	newstest2013	Params
128	21.50 ± 0.16 (21.66)	36.13M
256	21.73 ± 0.09 (21.85)	46.20M
512	21.78 ± 0.05 (21.83)	66.32M
1024	21.36 ± 0.27 (21.67)	106.58M
2048	$\mathbf{21.86} \pm 0.17$ (22.08)	187.09M

Table 1: BLEU scores on newstest2013, varying the embedding dimensionality.

4.2 RNN Cell Variant

Both LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Cho et al., 2014) cells are commonly used in NMT architectures. While there exist studies (Greff et al., 2016) that explore cell variants on small sequence tasks of a few thousand examples, we are not aware of such studies in large-scale NMT settings.

A motivation for gated cells such as the GRU and LSTM is the vanishing gradient problem. Using vanilla RNN cells, deep networks cannot efficiently propagate information and gradients through multiple layers and time steps. However, with an attention-based model, we believe that the decoder should be able to make decisions almost exclusively based on the current input and the attention context and we hypothesize that the gating mechanism in the decoder is not strictly necessary. This hypothesis is supported by the fact that we always initialize the decoder state to zero instead of passing the encoder state, meaning that the decoder state does not contain information about the encoded source. We test our hypothesis by using a vanilla RNN cell in the decoder only (Vanilla-Dec below). For the LSTM and GRU variants we replace cells in both the encoder and decoder. We use LSTM cells without peephole connections and initialize the forget bias of both LSTM and GRU cells to 1.

Cell	newstest2013	Params
LSTM	$\mathbf{22.22} \pm 0.08$ (22.33)	68.95M
GRU	21.78 ± 0.05 (21.83)	66.32M
Vanilla-Dec	15.38 ± 0.28 (15.73)	63.18M

Table 2: BLEU scores on newstest2013, varying the type of encoder and decoder cell.

In our experiments, *LSTM cells consistently outperformed GRU cells*. Since the computational bottleneck in our architecture is the softmax operation we did not observe large difference in training speed between LSTM and GRU cells. Somewhat to our surprise, we found that the vanilla decoder is unable to learn nearly as well as the gated variant. This suggests that the decoder indeed passes information in its own state throughout multiple time steps instead of relying solely on the attention mechanism and current input (which includes the previous attention context). It could also be the case that the gating mechanism is necessary to mask out irrelevant parts of the inputs.

4.3 Encoder and Decoder Depth

We generally expect deeper networks to converge to better solutions than shallower ones (He et al., 2016). While some work (Luong et al., 2015b;

Zhou et al., 2016; Luong and Manning, 2016; Wu et al., 2016) has achieved state of the art results using deep networks, others (Jean et al., 2015; Chung et al., 2016; Sennrich et al., 2016b) have achieved similar results with far shallower ones. Hence, it is unclear how important depth is, and whether shallow networks are capable of producing results competitive with those of deep networks. Here, we explore the effect of both encoder and decoder depth up to 8 layers. For the bidirectional encoder, we separately stack the RNNs in both directions. For example, the Enc-8 model corresponds to one forward and one backward 4-layer RNN. For deeper networks, we also experiment with two variants of residual connections (He et al., 2016) to encourage gradient flow. In the standard variant, shown in equation (4), we insert residual connections between consecutive layers. If $h_t^{(l)}(x_t^{(l)}, h_{t-1}^{(l)})$ is the RNN output of layer l at time step t, then:

$$x_t^{(l+1)} = h_t^{(l)}(x_t^{(l)}, h_{t-1}^{(l)}) + x_t^{(l)} \qquad (4)$$

where $x_t^{(0)}$ are the embedded input tokens. We also explore a dense ("ResD" below) variant of residual connections similar to those used by (Huang et al., 2016a) in Image Recognition. In this variant, we add skip connections from each layer to all other layers:

$$x_t^{(l+1)} = h_t^{(l)}(x_t^{(l)}, h_{t-1}^{(l)}) + \sum_{j=0}^{l} x_t^{(j)} \qquad (5)$$

Our implementation differs from (Huang et al., 2016a) in that we use an addition instead of a concatenation operation in order to keep the state size constant.

Table 3 shows results of varying encoder and decoder depth with and without residual connection. We found no clear evidence that encoder depth beyond two layers is necessary, but found deeper models with residual connections to be significantly more likely to diverge during training. The best deep residual models achieved good results, but only one of four runs converged, as suggested by the large standard deviation.

On the decoder side, deeper models outperformed shallower ones by a small margin, and we found that without residual connections, it was impossible for us to train decoders with 8
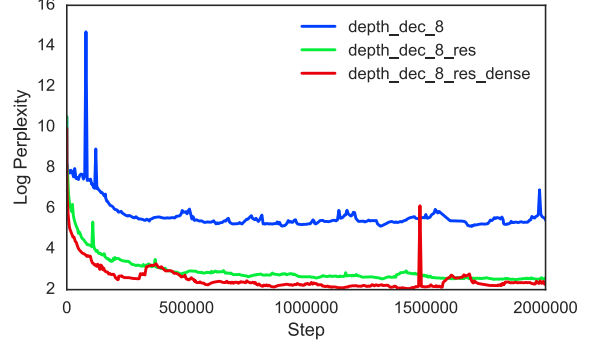

Figure 2: Training plots for deep decoder with and without residual connections, showing log perplexity on the eval set.

Depth	newstest2013	Params
Enc-2	21.78 ± 0.05 (21.83)	66.32M
Enc-4	$\mathbf{21.85} \pm 0.32$ (22.23)	69.47M
Enc-8	21.32 ± 0.14 (21.51)	75.77M
Enc-8-Res	19.23 ± 1.96 (21.97)	75.77M
Enc-8-ResD	17.30 ± 2.64 (21.03)	75.77M
Dec-1	21.76 ± 0.12 (21.93)	64.75M
Dec-2	21.78 ± 0.05 (21.83)	66.32M
Dec-4	$\mathbf{22.37} \pm 0.10$ (22.51)	69.47M
Dec-4-Res	17.48 ± 0.25 (17.82)	68.69M
Dec-4-ResD	21.10 ± 0.24 (21.43)	68.69M
Dec-8	01.42 ± 0.23 (1.66)	75.77M
Dec-8-Res	16.99 ± 0.42 (17.47)	75.77M
Dec-8-ResD	20.97 ± 0.34 (21.42)	75.77M

Table 3: BLEU scores on newstest2013, varying the encoder and decoder depth and type of residual connections.

or more layers. Across the deep decoder experiments, dense residual connections consistently outperformed regular residual connections and converged much faster in terms of step count, as shown in figure 2. We expected deep models to perform better (Zhou et al., 2016; Szegedy et al., 2015) across the board, and we believe that our experiments demonstrate the need for more robust techniques for optimizing deep sequential models. For example, we may need a better-tuned SGD optimizer or some form of batch normalization, in order to robustly train deep networks with residual connections.

4.4 Unidirectional vs. Bidirectional Encoder

In the literature, we see bidirectional encoders (Bahdanau et al., 2015), unidirectional encoders

(Luong et al., 2015a), and a mix of both (Wu et al., 2016) being used. Bidirectional encoders are able to create representations that take into account both past and future inputs, while unidirectional encoders can only take past inputs into account. The benefit of unidirectional encoders is that their computation can be easily parallelized on GPUs, allowing them to run faster than their bidirectional counterparts. We are not aware of any studies that explore the necessity of bidirectionality. In this set of experiments, we explore unidirectional encoders of varying depth with and without reversed source inputs, as this is a commonly used trick that allows the encoder to create richer representations for earlier words. Given that errors on the decoder side can easily cascade, the correctness of early words has disproportionate impact.

Cell	newstest2013	Params
Bidi-2	**21.78** ± 0.05 (21.83)	66.32M
Uni-1	20.54 ± 0.16 (20.73)	63.44M
Uni-1R	21.16 ± 0.35 (21.64)	63.44M
Uni-2	20.98 ± 0.10 (21.07)	65.01M
Uni-2R	21.76 ± 0.21 (21.93)	65.01M
Uni-4	21.47 ± 0.22 (21.70)	68.16M
Uni-4R	21.32 ± 0.42 (21.89)	68.16M

Table 4: BLEU scores on newstest2013, varying the type of encoder. The "R" suffix indicates a reversed source sequence.

Table 4 shows that bidirectional encoders generally outperform unidirectional encoders, but not by a large margin. The encoders with reversed source consistently outperform their non-reversed counterparts, but do not beat shallower bidirectional encoders.

4.5 Attention Mechanism

The two most commonly used attention mechanisms are the additive (Bahdanau et al., 2015) variant, equation (6) below, and the computationally less expensive multiplicative variant (Luong et al., 2015a), equation (7) below. Given an attention key h_j (an encoder state) and attention query s_i (a decoder state), the attention score for each pair is calculated as follows:

$$\text{score}(h_j, s_i) = \langle v, tanh(W_1 h_j + W_2 s_i)\rangle \quad (6)$$
$$\text{score}(h_j, s_i) = \langle W_1 h_j, W_2 s_i\rangle \quad (7)$$

We call the dimensionality of $W_1 h_j$ and $W_2 s_i$ the "attention dimensionality" and vary it from 128 to 1024 by changing the layer size. We also experiment with using no attention mechanism by initializing the decoder state with the last encoder state (None-State), or concatenating the last decoder state to each decoder input (None-Input). The results are shown in Table 5.

Attention	newstest2013	Params
Mul-128	22.03 ± 0.08 (22.14)	65.73M
Mul-256	22.33 ± 0.28 (22.64)	65.93M
Mul-512	21.78 ± 0.05 (21.83)	66.32M
Mul-1024	18.22 ± 0.03 (18.26)	67.11M
Add-128	22.23 ± 0.11 (22.38)	65.73M
Add-256	22.33 ± 0.04 (22.39)	65.93M
Add-512	**22.47** ± 0.27 (22.79)	66.33M
Add-1028	22.10 ± 0.18 (22.36)	67.11M
None-State	9.98 ± 0.28 (10.25)	64.23M
None-Input	11.57 ± 0.30 (11.85)	64.49M

Table 5: BLEU scores on newstest2013, varying the type of attention mechanism.

We found that the parameterized additive attention mechanism slightly but consistently outperformed the multiplicative one, with the attention dimensionality having little effect.

While we did expect the attention-based models to significantly outperform those without an attention mechanism, we were surprised by just how poorly the "Non-Input" models fared, given that they had access to encoder information at each time step. Furthermore, we found that the attention-based models exhibited significantly larger gradient updates to decoder states throughout training. This suggests that the attention mechanism acts more like a "weighted skip connection" that optimizes gradient flow than like a "memory" that allows the encoder to access source states, as is commonly stated in the literature. We believe that further research in this direction is necessary to shed light on the role of the attention mechanism and whether it may be purely a vehicle for easier optimization.

4.6 Beam Search Strategies

Beam Search is a commonly used technique to find target sequences that maximize some scoring function $s(\mathbf{y}, \mathbf{x})$ through tree search. In the simplest case, the score to be maximized is the log probability of the target sequence given the source.

Recently, extensions such as coverage penalties (Tu et al., 2016) and length normalizations (Wu et al., 2016) have been shown to improve decoding results. It has also been observed (Tu et al., 2017) that very large beam sizes, even with length penalty, perform worse than smaller ones. Thus, choosing the correct beam width can be crucial to achieving the best results.

Beam	newstest2013	Params
B1	20.66 ± 0.31 (21.08)	66.32M
B3	21.55 ± 0.26 (21.94)	66.32M
B5	21.60 ± 0.28 (22.03)	66.32M
B10	21.57 ± 0.26 (21.91)	66.32M
B25	21.47 ± 0.30 (21.77)	66.32M
B100	21.10 ± 0.31 (21.39)	66.32M
B10-LP-0.5	21.71 ± 0.25 (22.04)	66.32M
B10-LP-1.0	**21.80** ± 0.25 (22.16)	66.32M

Table 6: BLEU scores on newstest2013, varying the beam width and adding length penalties (LP).

Table 6 shows the effect of varying beam widths and adding length normalization penalties. A beam width of 1 corresponds to greedy search. We found that a well-tuned beam search is crucial to achieving good results, and that it leads to consistent gains of more than one BLEU point. Similar to (Tu et al., 2017) we found that very large beams yield worse results and that there is a "sweet spot" of optimal beam width. We believe that further research into the robustness of hyperparameters in beam search is crucial to progress in NMT. We also experimented with a coverage penalty, but found no additional gain over a sufficiently large length penalty.

4.7 Final System Comparison

Finally, we compare our best performing model across all experiments (base model with 512-dimensional additive attention), as chosen on the newstest2013 validation set, to historical results found in the literature in Table 8. While not the focus on this work, we were able to achieve further improvements by combining all of our insights into a single model described in Table 7.

Although we do not offer architectural innovations, we do show that through careful hyperparameter tuning and good initialization, it is possible to achieve state of the art performance on standard WMT benchmarks. Our model is outperformed only by (Wu et al., 2016), a model which

Hyperparameter	Value
embedding dim	512
rnn cell variant	LSTMCell
encoder depth	4
decoder depth	4
attention dim	512
attention type	Bahdanau
encoder	bidirectional
beam size	10
length penalty	1.0

Table 7: Hyperparameter settings for our final combined model, consisting of all of the individually optimized values.

is significantly more complex and lacks a public implementation.

Model	newstest14	newstest15
Ours (experimental)	22.03	24.75
Ours (combined)	22.19	25.23
OpenNMT	19.34	-
Luong	20.9	-
BPE-Char	21.5	23.9
BPE	-	20.5
RNNSearch-LV	19.4	-
RNNSearch	-	16.5
Deep-Att[*]	20.6	-
GNMT[*]	24.61	-
Deep-Conv[*]	-	24.3

Table 8: Comparison to RNNSearch (Jean et al., 2015), RNNSearch-LV (Jean et al., 2015), BPE (Sennrich et al., 2016b), BPE-Char (Chung et al., 2016), Deep-Att (Zhou et al., 2016), Luong (Luong et al., 2015a), Deep-Conv (Gehring et al., 2016), GNMT (Wu et al., 2016), and OpenNMT (Klein et al., 2017). Systems with an [*] do not have a public implementation.

5 Open Source Release

We demonstrated empirically how small changes to hyperparameter values and different initialization can affect results, and how seemingly trivial factors such as a well-tuned beam search are crucial. To move towards reproducible research, we believe it is important that researchers start building upon common frameworks and data processing pipelines. With this goal in mind, we specifically built a modular software framework

that allows researchers to explore novel architectures with minimal code changes, and define experimental parameters in a reproducible manner. While our initial experiments are in Machine Translation, our framework can easily be adapted to problems in Summarization, Conversational Modeling or Image-To-Text. Systems such as OpenNMT (Klein et al., 2017) share similar goals, but do not yet achieve state of the art results (see Table 8) and lack what we believe to be crucial features, such as distributed training support. We hope that by open sourcing our experimental toolkit, we enable the field to make more rapid progress in the future.

All of our code is freely available at https://github.com/google/seq2seq/.

6 Conclusion

We conducted what we believe to be the first large-scale analysis of architecture variations for Neural Machine Translation, teasing apart the key factors to achieving state of the art results. We demonstrated a number of surprising insights, including the fact that beam search tuning is just as crucial as most architectural variations, and that with current optimization techniques deep models do not always outperform shallow ones. Here, we summarize our practical findings:

- Large embeddings with 2048 dimensions achieved the best results, but only by a small margin. Even small embeddings with 128 dimensions seem to have sufficient capacity to capture most of the necessary semantic information.

- LSTM Cells consistently outperformed GRU Cells.

- Bidirectional encoders with 2 to 4 layers performed best. Deeper encoders were significantly more unstable to train, but show potential if they can be optimized well.

- Deep 4-layer decoders slightly outperformed shallower decoders. Residual connections were necessary to train decoders with 8 layers and dense residual connections offer additional robustness.

- Parameterized additive attention yielded the overall best results.

- A well-tuned beam search with length penalty is crucial. Beam widths of 5 to 10 together with a length penalty of 1.0 seemed to work well.

We highlighted several important research questions, including the efficient use of embedding parameters (4.1), the role of attention mechanisms as weighted skip connections (4.5) as opposed to memory units, the need for better optimization methods for deep recurrent networks (4.3), and the need for a better beam search (4.6) robust to hyperparameter variations.

In addition, we release to the public an open source NMT framework specifically built to explore architectural innovations and generate reproducible experiments, along with configuration files for all our experiments.

References

Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *OSDI*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

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 *EMNLP*.

Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio. 2016. A character-level decoder without explicit segmentation for neural machine translation. In *ACL*.

Jonas Gehring, Michael Auli, David Grangier, and Yann N. Dauphin. 2016. A convolutional encoder model for neural machine translation. *CoRR* abs/1611.02344.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. 2016. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems* PP(99):1–11.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*.

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

Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. 2016a. Densely connected convolutional networks. *CoRR* abs/1608.06993.

Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. 2016b. Speed/accuracy trade-offs for modern convolutional object detectors. *CoRR* abs/1611.10012.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On using very large target vocabulary for neural machine translation. In *ACL*.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *EMNLP*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. OpenNMT: Open-source toolkit for neural machine translation. *CoRR* abs/1701.02810.

Minh-Thang Luong and Christopher D. Manning. 2016. Achieving open vocabulary neural machine translation with hybrid word-character models. In *ACL*.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015a. Effective approaches to attention-based neural machine translation. In *EMNLP*.

Minh-Thang Luong, Ilya Sutskever, Quoc V. Le, Oriol Vinyals, and Wojciech Zaremba. 2015b. Addressing the rare word problem in neural machine translation. In *ACL*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Edinburgh neural machine translation systems for wmt 16. In *ACL*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Neural machine translation of rare words with subword units. In *ACL*.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *CVPR*.

Zhaopeng Tu, Yang Liu, Lifeng Shang, Xiaohua Liu, and Hang Li. 2017. Neural machine translation with reconstruction. In *AAAI*.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation. In *ACL*.

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.

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