Decoupling Encoder and Decoder Networks for Abstractive Document Summarization

Ying Xu[1], **Jey Han Lau**[2,3], **Timothy Baldwin**[3] and **Trevor Cohn**[3]

[1]Monash University
[2]IBM Research
[3]The University of Melbourne
ying.xu@monash.edu, jeyhan.lau@gmail.com
tb@ldwin.net, trevor.cohn@unimelb.edu.au

Abstract

Abstractive document summarization seeks to automatically generate a summary for a document, based on some abstract "understanding" of the original document. State-of-the-art techniques traditionally use attentive encoder–decoder architectures. However, due to the large number of parameters in these models, they require large training datasets and long training times. In this paper, we propose decoupling the encoder and decoder networks, and training them separately. We encode documents using an unsupervised document encoder, and then feed the document vector to a recurrent neural network decoder. With this decoupled architecture, we decrease the number of parameters in the decoder substantially, and shorten its training time. Experiments show that the decoupled model achieves comparable performance with state-of-the-art models for in-domain documents, but less well for out-of-domain documents.

1 Introduction

Abstractive document summarization is a challenging natural language understanding task. Abstractive methods first encode the original document into a high-level representation, and then decode it into the target summary.

Rush et al. (2015) proposed the task of headline generation as the first step towards abstractive summarization. Instead of using the full document, the authors experimented with using the first sentence as input, with the aim of generating a coherent headline given the sentence.

The current state-of-art system for the task is based on an attentive encoder and a recurrent decoder (Chopra et al., 2016), which is an extension of the methodology of Rush et al. (2015). The encoder and decoder are trained jointly, and the decoder attends to different parts of the document during generation. It has a large number of parameters, and thus requires large-scale training data and long training times.

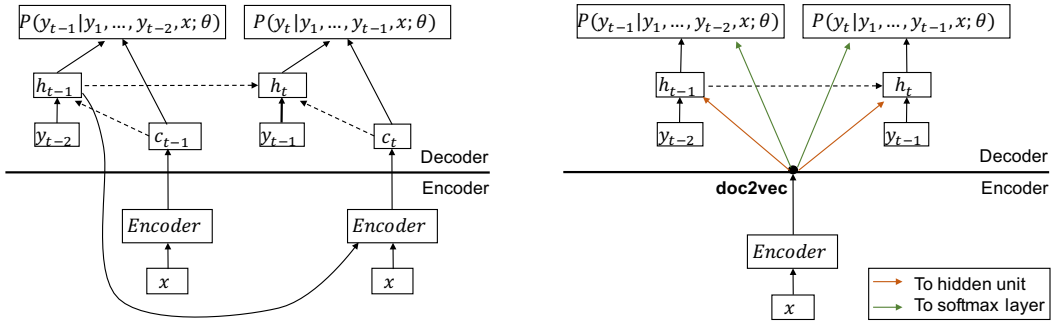
In this paper, we propose decoupling the encoder and decoder. We encode documents using `doc2vec` (Le and Mikolov, 2014), as it has been demonstrated to be a competitive unsupervised document encoder (Lau and Baldwin, 2016). We incorporate `doc2vec` vectors of input documents to the decoder as an additional signal, to generate sentences that are not only coherent but are also related to the original documents. Compared to the standard joint encoder–decoder design, the decoupled architecture has less parameters for the decoder, and thus requires less training data and trains faster.[1] The downside of the decoupled architecture is that the `doc2vec` signal is not updated in the decoder, and its document representation could be sub-optimal for the decoder to generate good summaries. Our experiments reveal that the decoupled architecture works well in-domain, but less well out-of-domain, as a consequence of the fixed capacity of the document encoding as well as having no explicit copy mechanism.

2 Attentive Recurrent Neural Network: A Joint Encoder–decoder Architecture

The attentive recurrent neural network is composed of an attentive encoder and a recurrent decoder (Chopra et al., 2016), where the encoder is

[1]The training time is decreased from 4 days (with full GIGAWORD) for the coupled model (Rush et al., 2015) to 2 days (with 75% GIGAWORD) in our model with comparable in-domain performance.

Proceedings of the MultiLing 2017 Workshop on Summarization and Summary Evaluation Across Source Types and Genres, pages 7–11,
Valencia, Spain, April 3, 2017.



(a) Joint encoder–decoder architecture

(b) Decoupled encoder–decoder architecture

Figure 1: Encoder–decoder architectures

a fixed-window feed-forward network and the decoder is a recurrent neural network (RNN: Elman (1998), Mikolov et al. (2010)) language model and parameters from both networks are trained together. Let $\mathbf{x}$ denote the document, $\mathbf{y}$ the summary, θ the set of parameters to be learnt, and $\mathbf{y} = y_1, y_2, ..., y_N$ a word sequence of length N. When decoding, $\mathbf{y}$ is computed as $\operatorname{argmax} P(\mathbf{y}|\mathbf{x};\theta)$, where the conditional probability $P(\mathbf{y}|\mathbf{x};\theta)$ can be calculated from each word y_t in the sequence, i.e. $P(\mathbf{y}|\mathbf{x};\theta) = \prod_{t=1}^{N} P(y_t|\{y_1,...,y_{t-1}\},\mathbf{x};\theta)$.

For the recurrent decoder, the conditional probability of each word is given as $P(y_t|\{y_1,...,y_{i-1}\},\mathbf{x};\theta) = g_\theta(\mathbf{h}_t,\mathbf{c}_t)$, where $\mathbf{h}_t$ represents the hidden state of RNN, i.e. $\mathbf{h}_t = g_\theta(y_{t-1},\mathbf{h}_{t-1},\mathbf{c}_t)$, and $\mathbf{c}_t$ the output of the encoder at time t.

For the attentive encoder, $\mathbf{x}$ is computed by attending to some of the source words using the previous hidden state $\mathbf{h}_{t-1}$.[2] Figure 1a demonstrates the dependencies between the next generated word y_t and document $\mathbf{x}$, given the parameters θ.

The decoupled architecture ensures that the information from document $\mathbf{x}$ is adapted to the current context of the generated summary.

3 Decoupled Encoder–decoder Architecture for Document Summarization

A decoupled encoder–decoder architecture has a clear boundary between the encoder and decoder: it can be seen as a pipeline model where the output of the encoder is fed as an input to the decoder, so the encoder and decoder modules can be trained separately. Figure 1b illustrates how the encoder and decoder are decoupled from each other. Here, we use `doc2vec` the document encoder and a long-short term memory network (LSTM: Hochreiter and Schmidhuber (1997)) as the decoder.

`doc2vec` is an extension of `word2vec` (Mikolov et al., 2013), a popular deep learning method for learning word embeddings. In `word2vec` (based on the `skipgram` variant) the embedding of a target word is learnt by optimising it to predict its position-indexed neighbouring words. `doc2vec` (based on the `dbow` variant) is based on the same idea, except that the target word is now the document itself, and the document vector is optimised to predict the document words. Note that the `dbow` implementation does not take into account the order of the words (hence its name "distributed bag of words"). Once the model is trained, embeddings of new/unseen documents can be inferred from the pre-trained model efficiently. As an encoder, `doc2vec` is completely unsupervised, and uses no labelled information or signal from the decoder.

The decoder is an RNN language model (Mikolov et al., 2010), implemented as an LSTM (Hochreiter and Schmidhuber, 1997). Formally:

$$\begin{aligned}
\mathbf{i}_t &= W_i\mathbf{x}_t + U_i\mathbf{h}_{t-1} + \mathbf{b}_i \\
\mathbf{f}_t &= W_f\mathbf{x}_t + U_f\mathbf{h}_{t-1} + \mathbf{b}_f \\
\mathbf{o}_t &= W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1} + \mathbf{b}_o \\
\mathbf{j}_t &= W_j\mathbf{x}_t + U_j\mathbf{h}_{t-1} + \mathbf{b}_j \\
\mathbf{c}_t &= \mathbf{c}_{t-1} * \sigma(\mathbf{f}_t) + \tanh(\mathbf{j}_t) * \sigma(\mathbf{i}_t) \\
\mathbf{h}_t &= \tanh(\mathbf{c}_t) * \sigma(\mathbf{o}_t)
\end{aligned} \tag{1}$$

where $\mathbf{i}_t$, $\mathbf{f}_t$ and $\mathbf{o}_t$ are input, forget and output gates, respectively; $\mathbf{j}_t$, $\mathbf{c}_t$ and $\mathbf{h}_t$ represent the new

[2] The unnormalised attention weights are computed by combining $\mathbf{h}_{t-1}$ with the convolutional embedding of each source word via dot product.

Combination	Equation
add-input	$\mathbf{i}'_t = \mathbf{i}_t + \mathbf{d}$
add-hidden	$\mathbf{h}'_t = \mathbf{h}_t + \mathbf{d}$
stack-input	$\mathbf{i}'_t = [\mathbf{i}_t; \mathbf{d}]$
stack-hidden	$\mathbf{h}'_t = [\mathbf{h}_t; \mathbf{d}]$
mlp-input	$\mathbf{i}'_t = \tanh(W_i \mathbf{i}_t + W_d \mathbf{d} + b)$
mlp-hidden	$\mathbf{h}'_t = \tanh(W_i \mathbf{h}_t + W_d \mathbf{d} + b)$

Table 1: Incorporation of doc2vec signal in the decoder. $\mathbf{d}$ denotes the doc2vec vector; $\mathbf{i}_t$ ($\mathbf{h}_t$) is the input (hidden) vector at time t; and "$[\cdot;\cdot]$" denotes vector concatenation.

input, new context and new hidden state, respectively; $*$ is the elementwise vector product; and σ is the sigmoid activation function.

Given an input word and previous hidden state, the decoder predicts the next word and generates the summary one word at a time.

To generate summaries that are related to the document, we incorporate the doc2vec input document signal to the decoder using several methods proposed by Hoang et al. (2016). There are two layers where we can incorporate doc2vec: in the input layer (input), or hidden layer (hidden). There are three methods of incorporation: addition (add), stacking (stack), or via a multilayer perceptron (mlp). Table 1 illustrates the 6 possible approaches to incorporation.

Note that add requires doc2vec to have the same vector dimensionality as the layer it is combined with, and stack-hidden doubles the hidden size (assuming they have the same dimensions), resulting in a large output projection matrix and longer training time.

4 Experiments and Results

4.1 Datasets

We test our decoupled architecture for the headline generation task. Following Chopra et al. (2016), we run experiments using GIGAWORD, DUC03 and DUC04.[3]

GIGAWORD is preprocessed according to Rush et al. (2015), yielding 4.3 million examples. For in-domain experiments, we randomly sample 2,000 examples for each validation and test set, and use the remaining examples for training. We tune hyper-parameters based on validation perplexity and evaluate performance on the test set using the ROUGE metric (Lin, 2004), following the same evaluation style of benchmark systems (Rush et al., 2015; Chopra et al., 2016). For out-of-domain experiments, we use the same models trained from GIGAWORD, but tune using DUC03 and test on DUC04; DUC03 and DUC04 each have 500 examples.

For the doc2vec encoder, we train using GIGAWORD and infer document vectors for validation and test examples using the trained model. Valid and test examples are excluded from the doc2vec training data.

4.2 Hyper-parameter tuning

For the encoder, we explore using a range of document lengths (first 20/30/40/50 words) to generate the input representation. Validation results show that using the first 20 words produces the best performance, suggesting that this length contains sufficient information to generate headlines.

We next test the 6 different ways to incorporate doc2vec into the decoder. We find that stacking the doc2vec vector with the input (stack-input) has the most consistent performance, while mlp is competitive, and add performs the worst. Interestingly, for mlp and stack, we find the difference between input and hidden to be small.

For the recurrent decoder, hyper-parameters that are tuned include the mini-batch size, hidden layer size, number of LSTM layers, number of training epochs, learning rate, and drop out rate. The best results is achieved with a mini-batch size of 128, hidden size of 900, and one LSTM layer. The best perplexity is obtained after 3 to 4 epochs, with a learning rate of 0.001. More training epochs are needed when we reduce the learning rate to 0.0001. For in-domain experiments, the best results are achieved with a dropout rate of 0.1, while for out-of-domain experiments, the best performance prefers a higher dropout rate at 0.4. This suggests that dropout plays an important role in combating over-fitting, and it is especially useful for out-of-domain data.

4.3 Results

We compare our model (RDS: Recurrent Decoupled Summarizer) with 4 state-of-art models: ABS, ABS+ (Rush et al., 2015); RAS-LSTM and RAS-Elman (Chopra et al., 2016), which are all joint encoder–decoder models. For in-domain results, we present ROUGE-1/2/L full-length F-scores in Table 2. For out-of-domain results we

[3] GIGAWORD: https://catalog.ldc.upenn.edu/LDC2003T05; DUC: http://duc.nist.gov/

System	ROUGE-1	ROUGE-2	ROUGE-L
ABS	29.6	11.3	26.4
ABS+	29.8	11.9	30.0
RAS-LSTM	32.6	14.7	30.0
RAS-Elman	33.8	16.0	31.2
RDS	*30.7*	*11.3*	*27.6*
RDS (75%)	29.1	10.0	26.3
RDS (50%)	27.4	8.9	24.9

Table 2: Comparison of ROUGE scores (full length F-score) for in-domain experiments.

System	ROUGE-1	ROUGE-2	ROUGE-L
Prefix	22.4	6.5	19.7
ABS	26.6	7.1	22.1
ABS+	28.2	8.5	23.8
RAS-LSTM	27.4	7.7	23.1
RAS-Elman	29.0	8.3	24.1
RDS	*16.7*	*3.7*	*14.4*

Table 3: Comparison of ROUGE scores (recall at 75 bytes) for out-of-domain experiments.

report ROUGE-1/2/L recall at 75 bytes in Table 3, where only the first 75 bytes of model-generated summary is used for evaluation against the references.

ABS employs an attentive encoder and a feed-forward neural network decoder. ABS+ works in the same way as ABS, but further tunes the decoder using Z-MERT (Zaidan, 2009). RAS-LSTM and RAS-Elman are detailed in Section 2; the only difference between them is that RAS-LSTM uses an LSTM decoder, while RAS-Elman uses a simple RNN decoder. Prefix is a baseline model where the first 75 byte of the document is used as the title.

Looking at Table 2, models with a recurrent decoder (RDS and RAS) perform better than those with a feed-forward decoder (ABS). The decoupled architecture RDS achieves competitive performance, although the fully joint RAS models achieve the best results. Chopra et al. (2016) found that RAS models perform best with a beam size of 10, while we found that RDS performs best with greedy argmax decoding.

We further experiment with training RDS using less data (50% and 75%), and find that its performance degrades slightly. Encouragingly, its ROUGE-1 is comparable to ABS when RDS is trained using only 75% of the training data, and it takes 2 days of training time (RDS) instead of 4 days (ABS).

I	A North Korean man **arrived in Seoul** Wednesday and **sought asylum** after escaping his hunger-stricken homeland, government officials said.
A	North Korean man **arrives in Seoul to seek asylum** for homeland security officials say.
D	North Korean man **defects to** South Korea.
I	**King Norodom Sihanouk** has **declined** requests to chair a summit of Cambodia 's top **political leaders**, saying the meeting would not bring any progress in deadlocked negotiations to form a government .
A	King Sihanouk **declines** to meet Cambodian leaders on eve of talks with Cambodia.
D	**Cambodian king refuses** to meet with leaders of **political leaders**.
I	Former U.S. president Jimmy Carter , who seems a perennial Nobel peace prize also-ran , **could have won** the coveted honor in 1978 **had it not been for strict deadline rules for nominations.**
A	Former U.S. president Jimmy Carter **wins** top honor for Nobel peace prize nominations.
D	Carter **wins** Nobel prize in literature.

Table 4: System generated article summaries. I: reference summary; A: RAS-Elman; and D: RDS.

For out-of-domain experiments, we compute ROUGE recall at 75 bytes and find that RDS performs poorly, even worse than the baseline method Prefix. This suggests that the decoupled architecture is sensitive to domain differences, and highlights a potential downside of the architecture. Investigating methods that can improve cross-domain performance is a direction for future work.

5 Discussion and Conclusion

We present some summaries for DUC documents generated by RAS-Elman and RDS in Table 4 to better understand possible reasons for the poor in-domain performance of RDS. The first observation is the shorter headlines generated by RDS compared to the DUC reference summaries and RAS-Elman headlines. RDS headlines are short — at an average length of 8.13 — and this is due to the short length of GIGAWORD titles it is trained on, at 8.50 words on average. On the other hand, the average length of DUC reference summaries and RAS-Elman headlines is 11.63 and 13.08 words respectively; this explains why RAS-Elman achieves better performance, since a longer sentence would have a better chance to score higher in ROUGE.

We also find that RDS often generates similar words and is penalised by ROUGE as it uses exact

word matching, as is evidenced by *Sihanouk* and *Cambodian king* in the second example.[4] Lastly, the bag-of-words view of the doc2vec encoder results in some meaning loss: in the third example, *Jimmy Carter* did not actually win the Nobel peace prize, for example.

We also computed the source copy rate of the systems.[5] We find that, on average, RDS copies only 50.7% of its predicted words from input document, while ABS and ABS+ copy at a rate of 85.4% and 91.5%. This is interesting, as it suggests that it paraphrases more than other systems, while achieving similar ROUGE performance.

To summarise, we proposed decoupling the encoder–decoder architecture as is traditionally used in sequence-to-sequence problems. We tested the decoupled system on news title generation, and found that it performed competitively in-domain. Out-of-domain experiments, however, reveal sub-par performance, suggesting that the decoupled architecture is susceptible to domain differences.

References

Sumit Chopra, Michael Auli, and Alexander M. Rush. 2016. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–98, San Diego, California, June. Association for Computational Linguistics.

Jeffrey Elman. 1998. Generalization, simple recurrent networks, and the emergence of structure. In *Proceedings of the Twentieth Annual Conference of the Cognitive Science Society*.

Cong Duy Vu Hoang, Trevor Cohn, and Gholamreza Haffari. 2016. Incorporating side information into recurrent neural network language models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1250–1255, San Diego, California, June. Association for Computational Linguistics.

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

Jey Han Lau and Timothy Baldwin. 2016. An empirical evaluation of doc2vec with practical insights into document embedding generation. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 78–86, Berlin, Germany, August. Association for Computational Linguistics.

Quoc V Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In Stan Szpakowicz Marie-Francine Moens, editor, *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81, Barcelona, Spain, July. Association for Computational Linguistics.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal, September. Association for Computational Linguistics.

Omar F. Zaidan. 2009. Z-MERT: A fully configurable open source tool for minimum error rate training of machine translation systems. *The Prague Bulletin of Mathematical Linguistics*, 91:79–88.

[4] Informal manual evaluation for RDS, ABS, ABS+ and RAS on GIGAWORD reveals that ABS, ABS+ and RAS are also generating words that are of similar meaning, and that overall, RDS is less prefered by annotators.

[5] Source copy rate is defined as the fraction of generated words that are in the original source documents.