Introduction to Neural Machine Translation with GPUs (part 3)

Share:
(https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/)

Posted on **July 26, 2015 (https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/)** by **Kyunghyun Cho (https://devblogs.nvidia.com/parallelforall/author/kcho/)** | **34 Comments (https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/#disqus_thread)**
Tagged **Deep Learning (https://devblogs.nvidia.com/parallelforall/tag/deep-learning/)**, **Machine Learning (https://devblogs.nvidia.com/parallelforall/tag/machine-learning/)**, **Machine Translation (https://devblogs.nvidia.com/parallelforall/tag/machine-translation/)**, **Natural Language Understanding (https://devblogs.nvidia.com/parallelforall/tag/natural-language-understanding/)**, **Neural Networks (https://devblogs.nvidia.com/parallelforall/tag/neural-networks/)**, **Theano (https://devblogs.nvidia.com/parallelforall/tag/theano/)**

Note: This is the final part of a detailed three-part series on machine translation with neural networks by Kyunghyun Cho. You may enjoy part 1 (http://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-with-gpus/) and part 2 (http://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/).

In the previous post in this series (http://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/), I introduced a simple encoder-decoder model for machine translation. This simple encoder-decoder model is excellent at English-French translation. However, in this post I will briefly discuss the weakness of this simple approach, and describe a recently proposed way of incorporating a soft attention mechanism to overcome the weakness and significantly improve the translation quality.

Furthermore, I will present some more recent works that utilize this neural machine translation approach to go beyond machine translation of text, such as image caption generation and video description generation. I'll finish the blog series with a brief discussion of future research directions and a pointer to the open source code implementing these neural machine translation models.

The Trouble with Simple Encoder-Decoder Architectures

In the encoder-decoder architecture, the encoder compresses the input sequence as a fixed-size vector from which the decoder needs to generate a full translation. In other words, the fixed-size vector, which I'll call a *context vector*, must contain every single detail of the source sentence. Intuitively, this means that the true function approximated by the encoder has to be extremely nonlinear and complicated. Furthermore, the dimensionality of the context vector must be large enough that a sentence of any length can be compressed.

In my paper "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches (http://arxiv.org/abs/1409.1259)" presented at SSST-8 (http://www.aclweb.org/anthology/W14-4000), my coauthors and I empirically confirmed that translation quality dramatically degrades as the length of the source sentence increases when the encoder-decoder model size is small. Together with a much better result from Sutskever et al. (2014) (http://arxiv.org/abs/1409.3215), using the same type of encoder-decoder architecture, this suggests that the representational power of the encoder needed to be large, which often means that the model must be large, in order to cope with long sentences (see Figure 1).

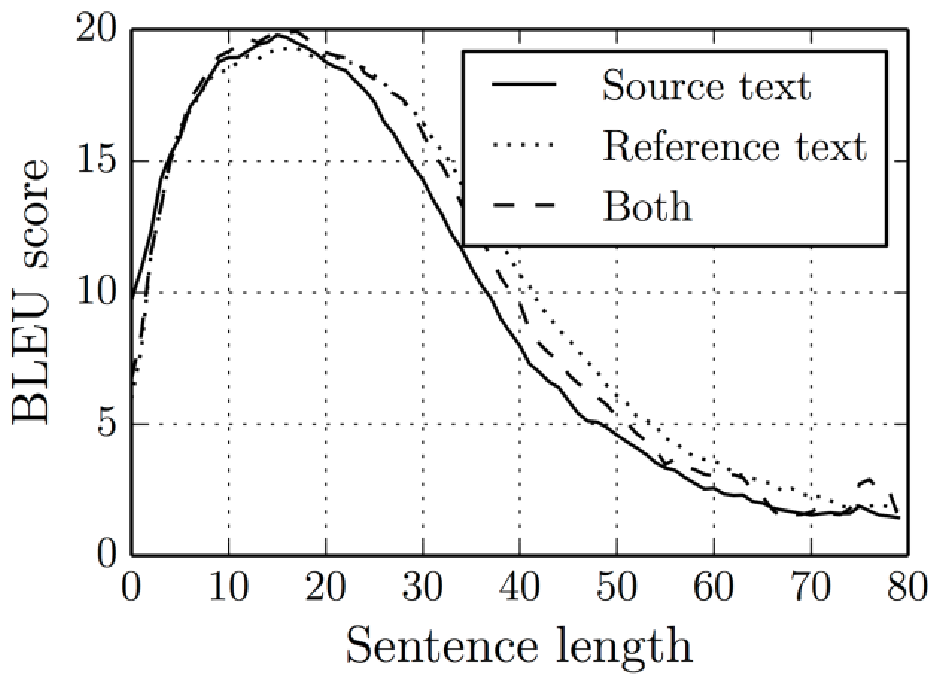


Figure 1: Dramatic drop of performance w.r.t. the length of sentence with a small encoder-decoder model.

Of course, a larger model implies higher computation and memory requirements. The use of advanced GPUs, such as NVIDIA Titan X, indeed helps with computation, but not with memory (at least not yet). The size of onboard memory is often limited to several Gigabytes, and this imposes a serious limitation on the size of the model. (Note: it's possible to overcome this issue by using multiple GPUs while distributing a single model across those GPUs, as shown by Sutskever et al. (2014 (http://arxiv.org/abs/1409.3215)). However, let's assume for now that we have access to a single machine with a single GPU due to space, power and other physical constraints.)

Then, the question is "can we do better than the simple encoder-decoder based model?"

Soft Attention Mechanism for Neural Machine Translation

The biggest issue with the simple encoder-decoder architecture is that a sentence of any length needs to be compressed into a fixed-size vector. This is a rather peculiar approach when you consider how computers typically handle compression tasks. When you zip a file, the length of the resulting compressed file is roughly proportional to the length of the original file. (This is not quite true: the compressed size is proportional to the *amount of information* in the original file, not its length. However, for the sake of argument, let us assume that the length of the original file closely reflects the amount of information in the file.)

Continuing with an analogy to digital computers, let's store the whole sentence *not* as a fixed-size vector, but as a memory that contains as many banks as there are source words. This is done by using a so-called bidirectional recurrent neural network (BiRNN) which consists of a forward recurrent neural network (RNN) and a separate *backward* RNN. As the names suggest, the forward and backward RNN's read the source sentence in forward and backward directions, respectively.

Now, let's call the hidden states from the forward RNN $\overrightarrow{h}_j$'s and those from the backward RNN $\overleftarrow{h}_j$'s. As we discussed in the first post of this series (http://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-with-gpus/), an RNN summarizes a sequence by reading one symbol at a time. This means that $\overrightarrow{h}_j$ of the forward RNN summarizes the source sentence up to the j-th word beginning from the first word, and $\overleftarrow{h}_j$ of the backward RNN up to the j-th word beginning from the *last* word. In other words, $\overrightarrow{h}_j$ and $\overleftarrow{h}_j$ together summarize the *whole* input sentence. See Figure 2 for an illustration.

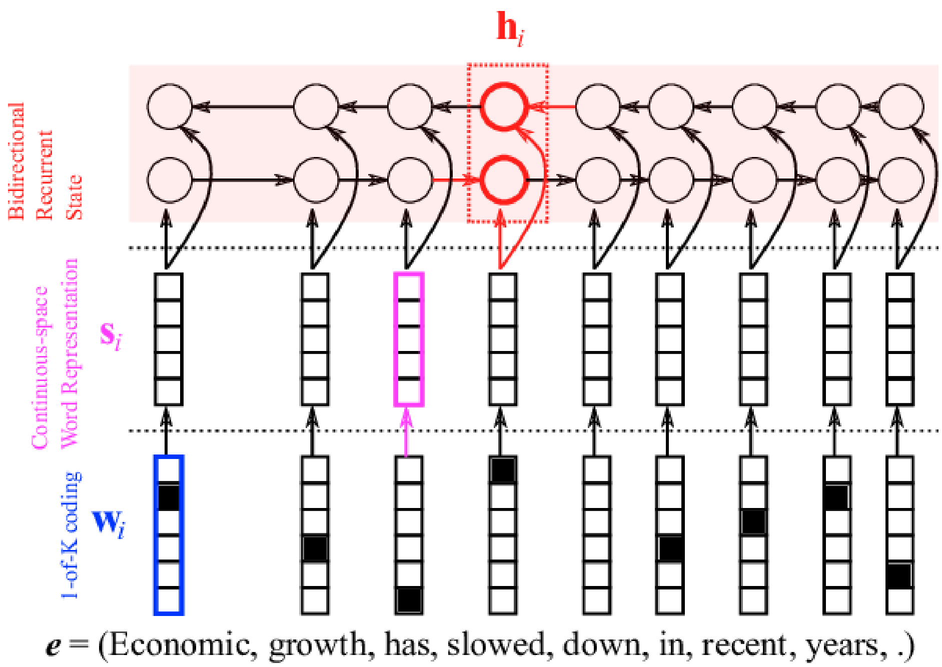


Figure 2. Bidirectional recurrent neural networks for encoding a source sentence.

This summary at the position of each word, however, is not the perfect summary of the whole input sentence. Due to its sequential nature, a recurrent neural network tends to remember recent symbols better. In other words, the further away an input symbol is from j, the less likely the RNN's hidden state, either $\overleftarrow{h}_j$ or $\overleftarrow{h}_j$, remembers it perfectly. The annotation vector, which we use to refer to the concatenation of $\overleftarrow{h}_j$ or $\overleftarrow{h}_j$, represents the current word w_j best.

This is definitely not an agreed convention, but personally, because of this reason, I understand the annotation vector as a *context-dependent word representation*. Furthermore, we can consider this set of context-dependent word representations as a mechanism by which we store the source sentence as a variable-length representation, as opposed to the fixed-length, fixed-dimensional summary from the simple encoder-decoder model.

With this variable-length representation of a source sentence, the decoder now needs to be able to *selectively* focus on one or more of the context-dependent word representations, or the annotation vectors, for each target word. So, which annotation vector should the decoder focus on each time?

Let's imagine you're translating the given source sentence, and you have written the first $i-1$ target words ($y_1, y_2, \ldots, y_{i-1}$) and are about to decide which target word you want to write as the i-th target word. In this case, how do you decide which source word(s) you will translate this time?

A typical translator looks at each source word x_j (or its context-dependent representation h_j), considers it together with the already translated words ($y_1, y_2, \ldots, y_{t-1}$) and decides whether the source word x_j has been translated (equivalently, how (ir)relevant the source word x_j is for the next target word). It repeats this process for every word in the source sentence.

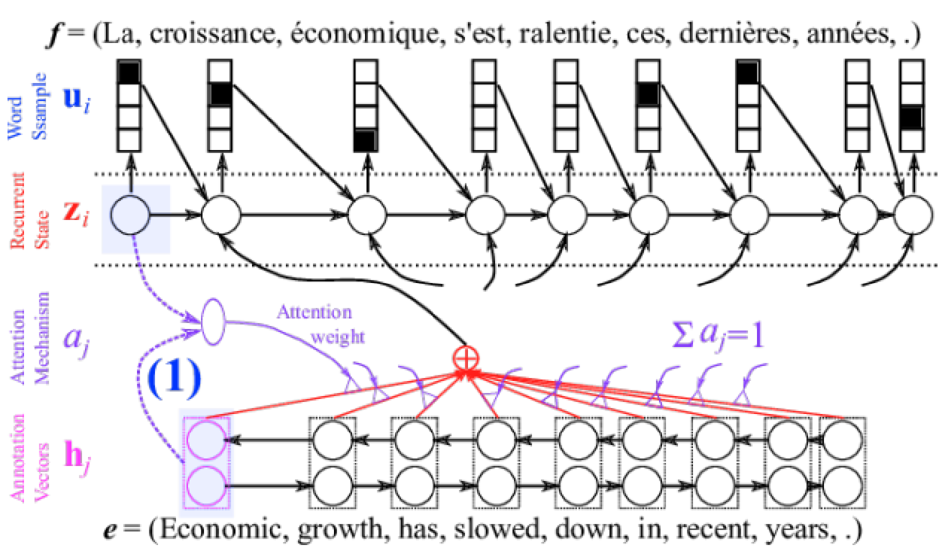


Figure 3. Attention Mechanism takes into consideration what has been translated and one of the source words.

Dzmitry Bahdanau and I together with Yoshua Bengio last summer (2014) proposed to include a small neural network in a decoder to do almost exactly this. The small neural network, which we call the *attention mechanism* (purple-colored part in Figure 3), takes as input the previous decoder's hidden state z_i (*what has been translated*) and one of the source context-dependent word representations h_j. The attention mechanism is implemented as a neural network with a single hidden layer and a single scalar output $e_j \in \mathbb{R}$, as in Figure 4. This is applied to every source word.

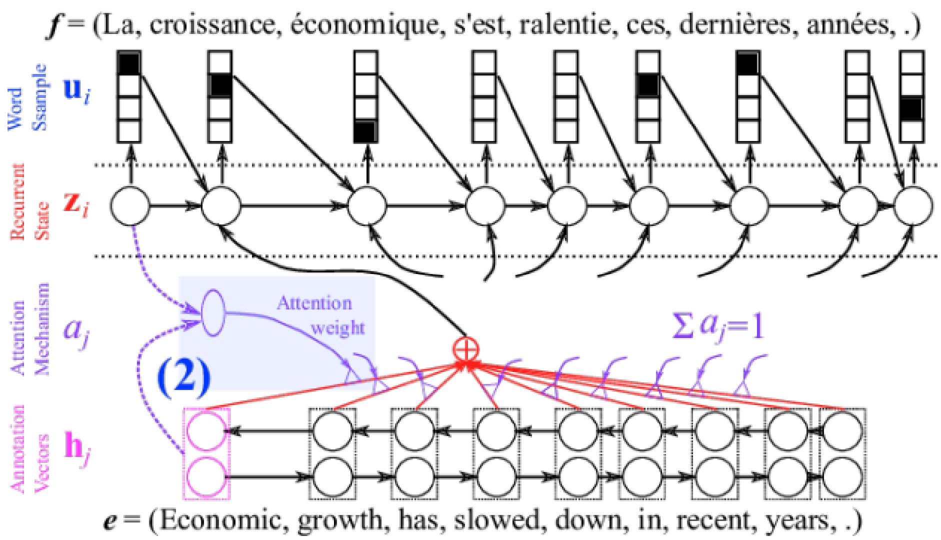


Figure 4. Attention Mechanism returns a single scalar corresponding to a relevance score of the j-th source word.

Once we've computed the relevance score of every source word, we want to make sure that they sum to one. This can be done easily using *softmax* normalization such that

$$\alpha_j = \frac{\exp(e_j)}{\sum_{j'} \exp(e_{j'})}.$$

See Figure 5 for the graphical illustration.

Why do we want this kind of normalization? There can be many reasons, but my favourite reason is that this helps us interpret the scores assigned by the attention mechanism in a probabilistic framework. From this probabilistic perspective, one can think of the attention weight α_j as the probability of the decoder selecting the j-th context-dependent source word representation out of all T source words. Then, we can compute the *expected* context-dependent word representation under this distribution (defined by the attention weights α_j) using

$$c_i = \sum_{j=1}^{T} \alpha_j h_j = \mathbb{E}_{\alpha'_j}\left[h_j\right].$$

This expected vector c_i summarizes the information about the whole source sentence, however, with different emphasis on different locations/words of the source sentence. Any annotation vector (context-dependent vector) deemed *relevant* (in other words, with high attention weight) by the attention mechanism will be better represented than those with low attention weights.

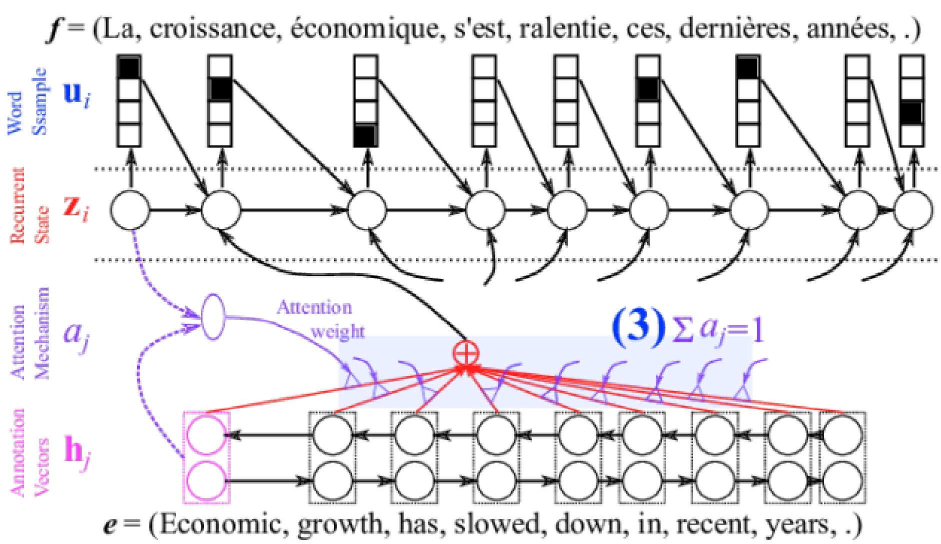


Figure 5. The relevance scores returned by the attention mechanism are normalized to sum to 1, which helps us interpret them as probabilities. From this probabilistic perspective, we compute the expectation of the annotation vectors under this distribution.

Once this vector c_i is computed, everything happens as we discussed in the decoder section of the previous post (http://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/), except that instead of h_T we use c_i at each time step i.

What does Soft Attention Mechanism Bring to the Table?

This approach of incorporating attention mechanism has become one of the hottest topics in deep learning recently (see Cho et al., 2015 (http://arxiv.org/abs/1507.01053) and references therein.) It would be super interesting to talk about the consequences of having attention mechanism in a neural network and how it can lead us to yet another level of success in deep learning, but that would take much more than a blog post. Instead, Figure 6 shows an example of the kind of attention (or alignment) that the model learns *without* any supervision on the alignment. (Note: although the term *weakly supervised* is often used to denote reinforcement learning, I find this type of model equally weakly supervised. Except for the final target translation, there was absolutely no supervision on the internal correspondence/attention/alignment.)

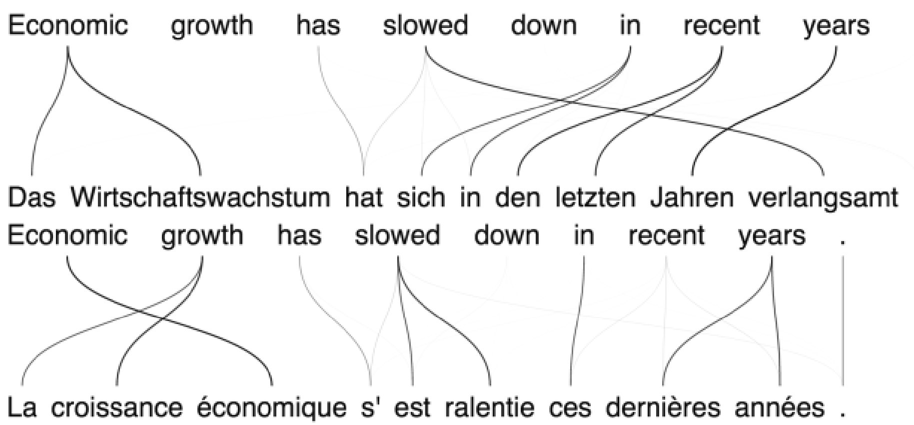


Figure 6. Sample translations made by the neural machine translation model with the soft-attention mechanism. Edge thicknesses represent the attention weights found by the attention model.

It's pretty awesome that the model automatically found the correspondence structure between two languages. I'm sure you'll appreciate it better if you speak either French or German (both of which I don't)! But, the bigger question is, does the introduction of the attention mechanism improve translation performance?

Yes, it does, and quite tremendously! Especially, in [Bahdandau et al., 2015] (http://arxiv.org/abs/1409.0473), we observed that with the addition of the attention mechanism, the quality of the translation does not drop as sentence length increases, even when the size of the model stays roughly the same, as Figure 7 shows.

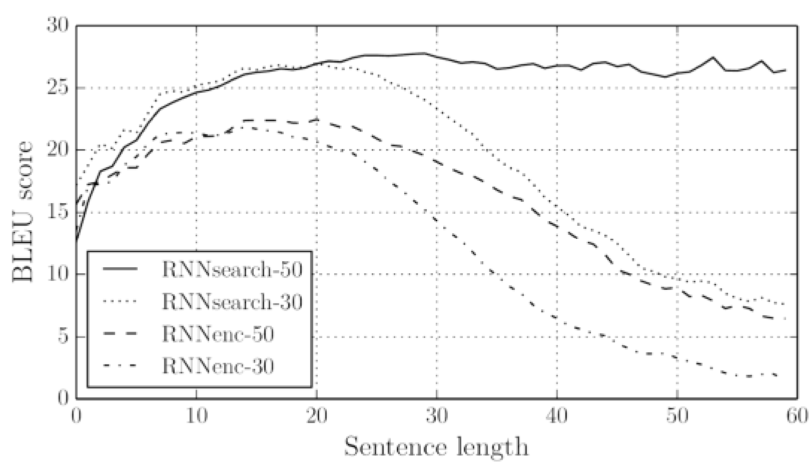


Figure 7. RNNsearch-50 is a neural machine translation model with the attention mechanism trained on all the sentence pairs of length at most 50.

The last issue is that this looks way too complicated to implement. Again, Theano to the rescue! Just write a forward computation pass (see here (https://github.com/kyunghyuncho/dl4mt-material/blob/master/session2/nmt.py#L464-L539) for an example) and use `theano.tensor.grad`, and you're ready to go.

NEURAL TURING MACHINES AND MEMORY NETWORKS

Reader Zzzz commented on my previous post in this series (http://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/#comment-2083041588):

> *Have you guys tried to use Neural Turing Machine or memory networks for this purpose? Or those models require much bigger training sets to produce better results?*

I meant to write a response sooner, but decided to wait until this post. Why? Because, both the neural Turing machine (NTM) by Graves et al. (2014) (http://arxiv.org/abs/1410.5401) as well as the memory networks by Weston et al. (2014) (http://arxiv.org/abs/1410.3916) can be thought of as variants/extensions of the described neural machine translation with soft attention mechanism. Think of the set of context-dependent word representations h_j as the contents in the memory, the attention mechanism as the read head of NTM and the decoder as the controller of NTM. These are very similar!

In fact, if you read the latest paper on memory networks by Sukhbaatar et al. (2015) (http://arxiv.org/abs/1503.08895), it becomes very clear that the attention-based neural machine translation, the NTM and the memory networks are more or less equivalent except for some details and for which applications they were designed. I, and probably you as well, have to wonder what will be the ultimate generalization of all these approaches and what kind of consequences this ultimate model will bring in the future.

Beyond translation: Image/Video Caption Generation

The most surprising and important point of this whole neural machine translation business is that *there's nothing specific to languages*. In particular, this approach can handle any type of input data as long as there's a suitable neural architecture that returns either a fixed-size vector representation of the input or a set of annotation vectors of it.

A recently published work from the University of Montreal and the University of Toronto showed that it is possible to design an attention-based encoder-decoder model which describes an image by replacing the encoder with a convolutional neural network, as Figure 8 shows. Similar approaches were proposed also in [Donahue et al., 2014; Fang et al., 2014; Karpathy and Li, 2014; Kiros et al., 2014; Mao et al., 2014].

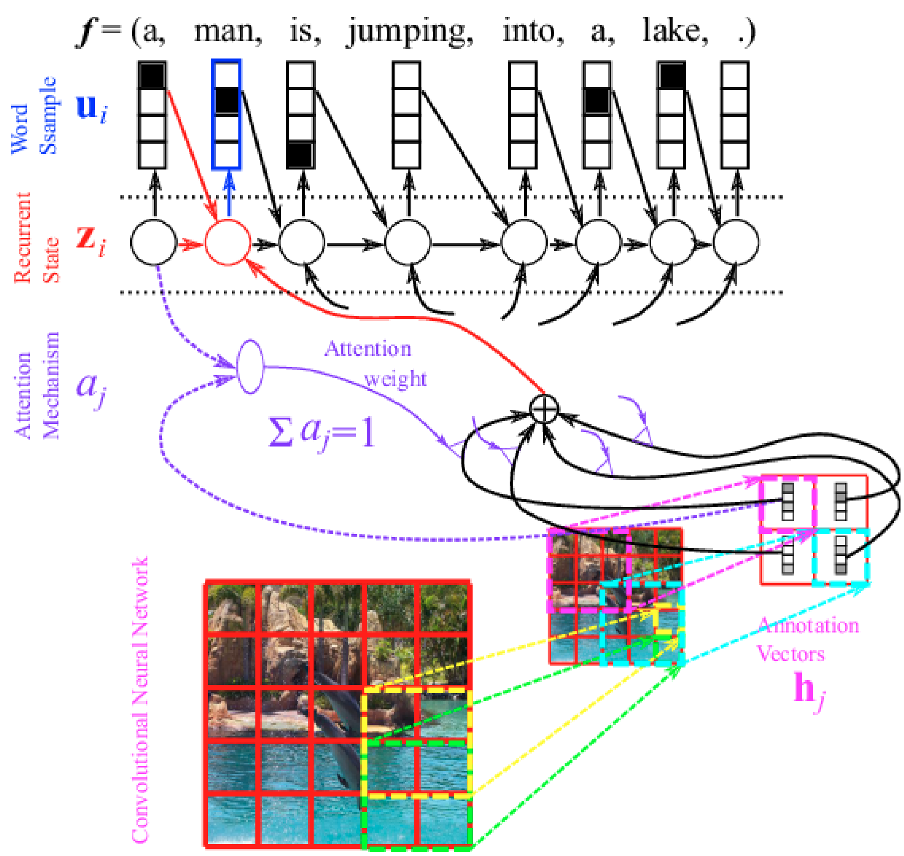


Figure 8. Image Caption Generation with Attention Mechanism.

Pushing the boundary further, Li et al. (2015) (http://arxiv.org/abs/1502.08029) applied a similar attention-based approach to video description generation by letting the decoder utilize temporal structures of the video. Similarly, video description generation with the simple encoder-decoder architecture was proposed recently by [Li et al., 2015; Venugopalan et al., 2015] (See Figure 9).

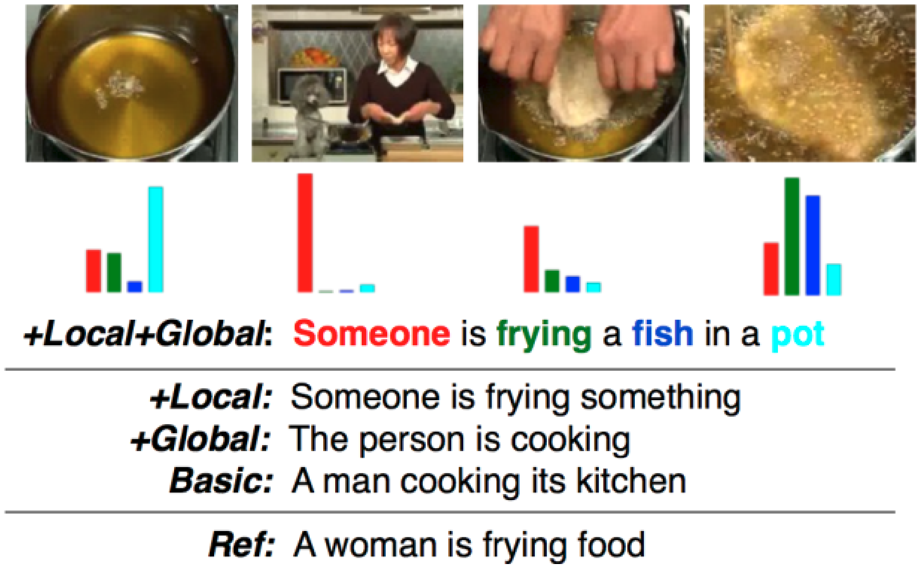


Figure 9. Temporal attention for video description generation. From [Li et al., 2015].

Furthermore, this same strategy of incorporating attention mechanism into learning to map from structured input to structured output has recently been applied to unimaginably many applications. One of my favorite is Vinyals et al. (2015) (http://arxiv.org/abs/1506.03134)'s application to discrete optimization, where they use this attention-based neural network to (approximately) solve the travelling salesperson problem! For the up-to-date extensive list of these applications, please refer to my recent overview paper (http://arxiv.org/abs/1507.01053).

Looking at all these recent works makes me wonder what will come next. What do you think? Let me know in the comments to this post.

What's Next?

In this post, I introduced recent advances in deep learning, centered around neural machine translation. However, there are many challenges related to neural machine translation remaining to be solved in the future. Let me list a few of them here.

- **Beyond modeling sentences:** Sentences, when represented as a sequence of words, are very short. The backpropagation algorithm for recurrent neural networks requires time proportional to the length of a sequence. Can there be another algorithm more suitable for dealing with much longer sequences, such as paragraphs and documents? Learning should probably be local, and weights should be updated online while processing a sequence.
- **Beyond natural languages:** Neural machine translation works for languages, but what else can it work for? Perhaps gene expression sequences, protein structure prediction, graphs and social networks, or weather data?
- **Multimodal learning**: Can we leverage other sources of information for translation? What is the natural way to incorporate multiple sources of information?

Where's the Code?

In this series so far I have avoided the discussion on the practical implementation. However, we at the University of Montreal have been actively making our code publicly available, because we believe open-source code helps speed up research progress by letting everyone try the code and easily customize or build upon it.

One of the most important contributions we have made in recent years has been Theano (http://deeplearning.net/software/theano/). Theano is a tool specialized in symbolically building an arbitrary neural network using Python. It abstracts the underlying compute architecture such that the same code can be run either on a more traditional CPU-based backend or on a more specialized hardware architecture such as GPUs. Furthermore, Theano implements symbolic differentiation which is crucial in training a neural network, making it extremely simple to quickly prototype complicated neural networks (such as the attention-based neural machine translation model described in this post). However, Theano only provides basic primitives, and one needs to sew those primitives into a neural network, which is not at all a trivial task even with Theano.

We (Razvan Pascanu, Caglar Gulcehre, myself, Dzmitry Bahdanau and Bart van Merrienboer) earlier released GroundHog (https://github.com/lisa-groundhog/GroundHog) which implements all the necessary components to build a neural machine translation model. Unfortunately, this code was rather hastily developed, and the readability is extremely low. Since then, Orhan Firat, Dzmitry Bahadanu and Bart van Merrienboer prepared an example script to build, train and use neural machine translation models based on the newly introduced framework called Blocks (https://github.com/mila-udem/blocks). However, this should also be treated only as an example for now.

In October this year (2015) at Dublin City University I will give a lecture on neural machine translation as part of the DL4MT Winter School (http://dl4mt.computing.dcu.ie/#/). To make the winter school more useful and practical for the audience, I am now preparing a stripped-down version of neural machine translation based on Theano. The code, which is called *dl4mt-material* and still in heavy preparation, can be found here (https://github.com/kyunghyuncho/dl4mt-material).

Currently, this code includes three subdirectories; `session0`, `session1` and `session2`. `session0` contains the implementation of the recurrent neural network language model (http://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-with-gpus/) using gated recurrent units, and `session1` the implementation of the simple neural machine translation model. In `session2`, you can find the implementation of the attention-based neural machine translation model we discussed today. I am planning to make a couple more sessions, so stay tuned!

Oh, one more thing before I finish this section, and the whole series. Let me make it clear again: *always train this type of model with GPUs!* To get a well-trained model, it takes about three to twelve days using a GeForce GTX Titan X, and with CPUs, I cannot even give any reasonable estimate on how long it will take to train a model. Of course, you can probably build a Google-sized cluster and use a distributed optimization algorithm, but I believe this is probably not the most cost-efficient way to go (unless you're Microsoft, Facebook or IBM).

Acknowledgements

I've covered a lot of recent research in this series of Parallel Forall posts on Neural Machine Translation. Hopefully I didn't make it sound like I did all this work on my own. Any work introduced here has been done by all the authors in the cited papers including those at the University of Montreal as well as other institutes. Furthermore, most of my work in the past few years wouldn't have been possible without the availability of Theano, and I'd like to thank all the contributors to Theano, and especially, Fred Bastien, Pascal Lamblin and Arnaud Bergeron.

Note that any error in the text is my own, and feel free to contact me if you found anything suspicious here.

REFERENCES

- Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).
- Bastien, Frédéric et al. "Theano: new features and speed improvements." arXiv preprint arXiv:1211.5590 (2012).
- Bergstra, James et al. "Theano: a CPU and GPU math expression compiler." Proceedings of the Python for scientific computing conference (SciPy) 30 Jun. 2010: 3.
- Bridle, J. S. (1990). Training Stochastic Model Recognition Algorithms as Networks can lead to Maximum Mutual Information Estimation of Parameters. In Touretzky, D., editor, Advances in Neural Information Processing Systems, volume 2, (Denver, 1989).
- Brown, Peter F et al. "The mathematics of statistical machine translation: Parameter estimation." Computational linguistics 19.2 (1993): 263-311.
- Cho, Kyunghyun et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." arXiv preprint arXiv:1406.1078 (2014).
- Cho, Kyunghyun, Aaron Courville, and Yoshua Bengio. "Describing Multimedia Content using Attention-based Encoder–Decoder Networks." arXiv preprint arXiv:1507.01053 (2015).
- Denil, Misha et al. "Learning where to attend with deep architectures for image tracking." Neural computation 24.8 (2012): 2151-2184.
- Donahue, Jeff et al. "Long-term recurrent convolutional networks for visual recognition and description." arXiv preprint arXiv:1411.4389 (2014).
- Fang, Hao et al. "From captions to visual concepts and back." arXiv preprint arXiv:1411.4952 (2014).
- Forcada, Mikel L, and Ñeco, Ramón P. "Recursive hetero-associative memories for translation." Biological and Artificial Computation: From Neuroscience to Technology (1997): 453-462.
- Graves, Alex, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines." arXiv preprint arXiv:1410.5401 (2014).
- Graves, Alex, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines." arXiv preprint arXiv:1410.5401 (2014).
- Gregor, Karol et al. "DRAW: A recurrent neural network for image generation." arXiv preprint arXiv:1502.04623 (2015).
- Gulcehre, Caglar et al. "On Using Monolingual Corpora in Neural Machine Translation." arXiv preprint arXiv:1503.03535 (2015).
- Kalchbrenner, Nal, and Phil Blunsom. "Recurrent Continuous Translation Models." EMNLP 2013: 1700-1709.
- Karpathy, Andrej, and Li, Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." arXiv preprint arXiv:1412.2306 (2014).
- Kingma, D. P., and Ba, J. "A Method for Stochastic Optimization." arXiv preprint arXiv:1412.6980 (2014).
- Kiros, Ryan, Ruslan Salakhutdinov, and Richard S Zemel. "Unifying visual-semantic embeddings with multimodal neural language models." arXiv preprint arXiv:1411.2539 (2014).
- Koehn, Philipp. Statistical machine translation. Cambridge University Press, 2009.

- Mao, Junhua et al. “Deep Captioning with Multimodal Recurrent Neural Networks (m-RNN).” arXiv preprint arXiv:1412.6632 (2014).
- Mnih, Volodymyr, Nicolas Heess, and Alex Graves. “Recurrent models of visual attention.” Advances in Neural Information Processing Systems 2014: 2204-2212.
- Pascanu, Razvan et al. “How to construct deep recurrent neural networks.” arXiv preprint arXiv:1312.6026 (2013).
- Schwenk, Holger. “Continuous space language models.” Computer Speech & Language 21.3 (2007): 492-518.
- Sukhbaatar, Sainbayar et al. “End-To-End Memory Networks.”
- Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. “Sequence to sequence learning with neural networks.” Advances in Neural Information Processing Systems 2014: 3104-3112.
- Venugopalan, Subhashini et al. “Sequence to Sequence–Video to Text.” arXiv preprint arXiv:1505.00487 (2015).
- Weston, Jason, Sumit Chopra, and Antoine Bordes. “Memory networks.” arXiv preprint arXiv:1410.3916 (2014).
- Xu, Kelvin et al. “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention.” arXiv preprint arXiv:1502.03044 (2015).
- Yao, Li et al. “Video description generation incorporating spatio-temporal features and a soft-attention mechanism.” arXiv preprint arXiv:1502.08029 (2015).
- Zeiler, Matthew D. “ADADELTA: an adaptive learning rate method.” arXiv preprint arXiv:1212.5701 (2012).

RELATED POSTS

Introduction to Neural Machine Translation with GPUs (part 1) (https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-with-gpus/)

Introduction to Neural Machine Translation with GPUs (Part 2) (https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/)

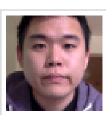
Deep Learning in a Nutshell: Core Concepts (https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-core-concepts/)

Understanding Aesthetics with Deep Learning (https://devblogs.nvidia.com/parallelforall/understanding-aesthetics-deep-learning/)

‖∀

Share:

About Kyunghyun Cho



Kyunghyun Cho is an assistant professor in the Department of Computer Science, Courant Institute of Mathematical Sciences and the Center for Data Science at New York University (NYU) (starting September, 2015). Previously, he was a postdoctoral researcher at the University of Montreal under the supervision of Prof. Yoshua Bengio after obtaining a doctorate degree at Aalto University (Finland) in early 2014. Kyunghyun's main research interests include neural networks, generative models and their applications, especially, to language understanding.

View all posts by Kyunghyun Cho → (https://devblogs.nvidia.com/parallelforall/author/kcho/)

Marcin Elantkowski • 2 years ago

Hey, thanks for the great post! NMT is one of the coolest recent applications of NNets out there.

I have two questions, however:
- Can Your code be used to train an actual working NMT model, or it's just a toy example for education purposes?
- In general, can a good NMT model be trained on a GTX (980) with 4Gibs of RAM, or these models need so many parameters they require a cluster of Titan Xs?

Edit:

I've got one more question.
When we use decoder to produce translation, it is obvious that we feed the output back to the input (like in character-level RNN language model).
During training however, what is the input to the decoder? Also it's own output, or rather the groundtruth translation? When training the char-RNN we input the true letters, not the sampled ones.

2 • Reply • Share ›

Kyunghyun Cho → Marcin Elantkowski • 2 years ago

Hi Marcin,

- "Can Your code be used to train an actual working NMT model, or it's just a toy example for education purposes?"

Yes, it can. It misses some post-processing routine for replacing unknown tokens and very large target vocabulary extension at the moment, but even without them, you can get a decent NMT model with the very code in those repos.

- "In general, can a good NMT model be trained on a GTX (980) with 4Gibs of RAM, or these models need so many parameters they require a cluster of Titan Xs?"

Yes, you can, but each model needs to be quite small and won't perform well. Instead, you can train multiple small models (each on GTX980) and make an ensemble of them.

- "During training however, what is the input to the decoder?"

It's the ground truth you feed in, because that is how it's supposed to be if the log-likelihood is maximized (check out the first post.) However, this does not mean that this is the only or best way. See, for instance, http://arxiv.org/abs/1506.0...

Cheers,
- K

• Reply • Share ›

Marcin Elantkowski → Kyunghyun Cho • 2 years ago

Thank You so much for the reply and the link.

It's too bad my new high-end GPU is still not good enough for deep learning, but I hope it'll suffice for educational purposes.

Thanks again,
Regards,
Marcin

• Reply • Share ›

Michael Joyner • 7 months ago

Has anyone implemented this or is willing to implement this in deeplearning4j?

I am looking for something to use as a starting point to try and translate Cherokee/English using a small corpus. (Only small corpus is available).

The combination of extreme language dissimilarity and small corpus does not work well with SMT.

1 • Reply • Share ›

KyungHyun Cho → Michael Joyner • 7 months ago

Not that I know of, but it'll be a great service to the community if someone implements it in deeplearning4j.

• Reply • Share ›

Michael Joyner → KyungHyun Cho • 7 months ago

Where would be a good place to ask for someone to do this? Assuming deeplearning4j is capable.

• Reply • Share ›

Adam Gibson → Michael Joyner • 7 months ago

Hi, Adam from deeplearning4j here. I haven't seen anyone doing machine translation work. We have a few NLP shops that write papers with us but none are doing translation. The new custom layer support should make this easier as well. We have seq2seq built in but not much beyond that. We haven't seen much in the way of machine translation work ourselves either.

Reply • Share ›

Michael Joyner ➦ Adam Gibson • 6 months ago

So, where does one go to ask for a 3rd party to do this kinda of stuff?

Reply • Share ›

Michael Joyner • 7 months ago

I see that y'all have authored a paper on uses non-aligned corpus materials.

Why not train a neural network to align paragraphs between two translations of the same document? (which would also indicate which paragraphs to treat as non-alignable).

Reply • Share ›

KyungHyun Cho ➦ Michael Joyner • 7 months ago

That is certainly an interesting problem, but may not necessarily require a neural net to do so. There have been quite a lot of work already on automatically aligning source and target sentences:

https://scholar.google.com/...

Reply • Share ›

Papar22 • 10 months ago

Many thanks for your nice post and illustration. I have one question regarding the possible dependencies in attention mechanism.

Have you ever tried other dependencies? I mean, for example, in your model, s(i) is dependent on c(i) or e(ij) is based on only s(i-1),not on the last two state. I am really interested to know have you tried other dependencies and you reached the conclusion that, this one works the best?

Is there any special meaning behind these dependencies?

Many thanks and sorry if the question is not relevant.

Reply • Share ›

KyungHyun Cho ➦ Papar22 • 7 months ago

I haven't tried it before, but certainly it is an active research topic. Some recent papers that I can think right away include (but definitely are not limited to):

http://arxiv.org/abs/1608.0...
http://arxiv.org/abs/1607.0...
http://arxiv.org/abs/1605.0...
http://papers.nips.cc/paper...

Many more such works have been presented at ACL, NAACL and EMNLP this year.

Reply • Share ›

Amr El-Desoky Mousa • a year ago

Hi,

Thanks so much for your very interesting illustrations. However, I wish you could answer my questions below.

1) In the "encoder" phase of the network, what are the targets used during training to bring up the fixed sentence representation? Do you use for example a fixed special word as a target to indicate "null" output at this phase, or you also train to produce the next word in sequence like in language modeling ? or something else? I am really very interested to know.

2) For the simpler model without attention mechanism, or even for the more complicated ones with the attention mechanism, is the encoder and decoder trained at the same time in one shot? or indeed they are two separate phases? I assume they are trained in one shot to optimize the log likelihood criterion that imposes the probability of target seq given the source seq, is that right?

3) When using the encoder-decoder network in translation mode (I mean not in training mode) in order to produce a target translation of a given source sentence.

We start by reading the source sentence sequentially until we produce its representation, then we get into the decoder phase, here the input that we need to feed in is a sample of the output, right? do you select highest probable word from the output here the input that we need to feed in is a sample of the output, right? do you select highest probable word from the output layer of the decoder? or you employ multiple N best words in a beam search strategy? please explain, and what is the stopping condition during this search?

see more

Reply • Share ›

Kyunghyun Cho Amr El-Desoky Mousa • a year ago

"In the "encoder" phase of the network, what are the targets used during training to bring up the fixed sentence representation?"

There is no target in the encoder network.

"For the simpler model without attention mechanism, or even for the more complicated ones with the attention mechanism, is the encoder and decoder trained at the same time in one shot?"

They are trained jointly without any pretraining. The separation between the encoder and decoder is rather conceptual.

"3)"

During training, you feed in the correct words from the training data set (as dictated by maximum likelihood estimation.) During test time, you feed in the word selected from the previous time step. In order to do better (approximate) decoding, it is usual to use beam search.

"can you please give an idea how much accuracy you got using your approaches compared to the best SMT and the best earlier NMT method?"

I suggest you to Table 1 of http://arxiv.org/abs/1507.0...

Reply • Share ›

Qian • a year ago

Thanks for very nice post!
I am very interested in your opinion about beyond modeling sentences. You said, "
Learning should probably be local, and weights should be updated online while processing a sequence."
I have a question, what's "weights should be updated online while processing a sequence"? Could you explain it more?

Reply • Share ›

Kyunghyun Cho Qian • a year ago

In the vanilla form of backpropagation through time (without truncation), one update of the weights requires reading a full sequence. If the sequence is really long (like 10s or 100s of thousands), it's probably not great to do this.

Reply • Share ›

hugman Sangkeun Jung • 2 years ago

Thanks for great post, and excellent codes.

I am trying to apply attention model to other NLP tasks.

What should I do If i want to specify or constraint number of target sequence in prediction? (say 1 or same number as input sequence)

Reply • Share ›

Kyunghyun Cho hugman Sangkeun Jung • a year ago

I suggest you to read Sec. 4.1.4 of my lecture note: http://arxiv.org/abs/1511.0...

Reply • Share ›

jason • 2 years ago

Thanks for the posts. I am trying to replicate your sessions from the git repo, the datasets you are using seems to be missing from the repo. Do you know where I can get these sets?

Thanks

Reply • Share ›

Kyunghyun Cho jason • a year ago

Please look at https://github.com/kyunghyu... which was written by Orhan Firat

Reply • Share ›

Wenfeng Xuan • 2 years ago

Thanks for the great post!
In figure 7, why RNNsearch-30 still suffer from the problem of performance degradation with respect to the sentence length?

Reply • Share ›

Kyunghyun Cho ➦ Wenfeng Xuan • a year ago

Unfortunately RNN is known to adapt to the lengths it has seen during training. This is perhaps the reason why this happens. For more interesting approaches to build an RNN that generalizes to sequences much longer than it has seen during training, see for instance http://papers.nips.cc/paper....

Reply • Share ›

Wenfeng Xuan ➦ Kyunghyun Cho • a year ago

Many thanks for your reply!

Reply • Share ›

Liling Tan • 2 years ago

Hi Kyunghyun, thanks for sharing the codes. I've some questions on these lines: https://github.com/kyunghyu... . What is `n_words_source` and what is the list comprehension and line 48 - 50 doing. Could you care to explain in brief?

Also, what does `source` at https://github.com/kyunghyu... refer to? Is that the list of frequencies of the words?

Once again, thanks for sharing the code. It's great to learn from the code and I've learnt much and still learning more from them! I wish everyone releases their code for any paper/tutorials no matter how "raw" they are. They are valuable documentation and great learning points!

Reply • Share ›

Kyunghyun Cho ➦ Liling Tan • 2 years ago

Hi Liling,

1. "What is `n_words_source`"

That is the maximum size of the source vocabulary. Any word with an index larger than this max size will be considered an unknown word (too rare.)

2. "what does `source` at https://github.com/kyunghyunch... refer to?"

that's a list of sentences in a minibatch.

Cheers,
- K

Reply • Share ›

Liling Tan ➦ Kyunghyun Cho • 2 years ago

HI Kyunghyun, thanks for the specific explanations to the code!!

Reply • Share ›

Xiang Li • 2 years ago

Very great post!
I have two questions.
1. We find NMT gets very good performance compared with SMT for only several years. But we know much linguistic knowledge play a key role in NLP and SMT. How do you think about gaining profit by integrating the linguistic knowledge into NMT?
2. Another question is that whether NMT can not limited by the expensive parallel corpus, but fully utilizing many comparable corpus? Because NMT models the source sentence as an abstractive representation, and doesn't actually model the word or phrase relationship between languages like SMT.

Reply • Share ›

Kyunghyun Cho ➦ Xiang Li • 2 years ago

Hi Xiang

Hi Xiang,

1. "How do you think about gaining profit by integrating the linguistic knowledge into NMT?"

I believe that the described NMT models already have captured those linguistic knowledge that's necessary for well performing translation. However, I also believe that the existing linguistic knowledge can be used as guiding signal during training. For instance, it may speed up the convergence of training by augmenting the encoder with additional classifiers or structured output predictors to predict certain linguistic properties of a source sentence that are deemed important. This kind of giving out hints to make learning easier has been tried in, e.g., http://arxiv.org/abs/1301.4083. So, not as input but as additional target.

2. "Another question is that whether NMT can not limited by the expensive parallel corpus, but fully utilizing many comparable corpus?"

Yes, I agree with you that it's important to utilize (almost infinite) monolingual corpora. However, it is not clear how it should be done in NMT. In fact, we here at Montreal jointly with our collaborators in France proposed one such way, called 'deep fusion', in http://arxiv.org/abs/1503.0.... Though, I need to tell you: this paper has been rejected twice already from ACL'15 and EMNLP'15.. :(

Cheers,
- K

1 ^ | v • Reply • Share ›

Marcin Elantkowski → Kyunghyun Cho • 2 years ago

Do You think that in the nearest future it will be possible to utilize "somewhat parallel" corpora, like translated books? Perhaps by forcing two language models to produce similar paragraph vectors or something like this?

^ | v • Reply • Share ›

Kyunghyun Cho → Marcin Elantkowski • 2 years ago

Well, this I have no answer to. Of course, I hope one day there will be no need for strict sentence/paragraph alignment, but it's quite unclear how it'll happen.

1 ^ | v • Reply • Share ›

Minglei LI • 2 years ago

Thank you! Very good post!!

"Furthermore, we can consider this set of context-dependent word representations as a mechanism by which we store the source sentence as a variable-length representation, as opposed to the fixed-length, fixed-dimensional summary from the simple encoder-decoder model."

1. Does that mean the representation length of the source sentence is determined by the number of annotation vector h and the length of h?

2. In addition, I think expected vector c_i can be seen as the final representations of the source sentence, whose length is fixed (the dimension of h). Is that right?

^ | v • Reply • Share ›

Kyunghyun Cho → Minglei LI • 2 years ago

Hi Minglei,

1. Ues and no, I meant that the length of the representation is proportional to the length of the source sentence, which is proportional to the number of annotation vectors.

2. Only at time i. The representation c_i changes w.r.t. i according to the attention weights which are determined by the decoder's hidden state (which again changes for each and every target word generated.)

Cheers
- K

^ | v • Reply • Share ›

Viacheslav Seledkin • a year ago

Hello! Thanks for great introduction!

I have a question:

What is the exact form of mixing "glimpses" generated by attention network and recurrent (GRU for example) decoding network?

Reply • Share ›

Kyunghyun Cho → Viacheslav Seledkin • a year ago

It's simply to concatenate the input, the previous hidden state and the context created by taking the weighted sum of the source annotation vectors (with the glimpses as the weights.)

Reply • Share ›