Eugenio Culurciello Follow
I dream and build new technology
Apr 13 · 8 min read

The fall of RNN / LSTM

We fell for Recurrent neural networks (RNN), Long-short term memory (LSTM), and all their variants. **Now it is time to drop them!**

It is the year 2014 and LSTM and RNN make a great come-back from the dead. We all read Colah's blog and Karpathy's ode to RNN. But we were all young and unexperienced. For a few years this was the way to solve sequence learning, sequence translation (seq2seq), which also resulted in amazing results in speech to text comprehension and the raise of Siri, Cortana, Google voice assistant, Alexa. Also let us not forget machine translation, which resulted in the ability to translate documents into different languages or neural machine translation, but also translate images into text, text into images, and captioning video, and … well you got the idea.

Then in the following years (2015–16) came ResNet and Attention. One could then better understand that LSTM were a clever bypass technique. Also attention showed that MLP network could be replaced by *averaging* networks influenced by a *context vector*. More on this later.

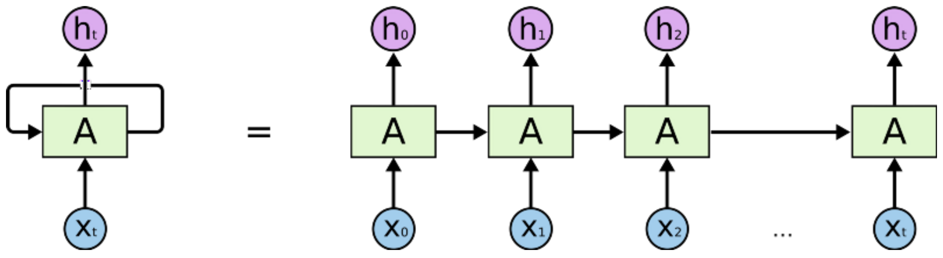
It only took 2 more years, but today we can definitely say:

"Drop your RNN and LSTM, they are no good!"

But do not take our words for it, also see evidence that Attention based networks are used more and more by Google, Facebook, Salesforce, to name a few. All these companies have replaced RNN and variants for attention based models, and it is just the beginning. RNN have the days counted in all applications, because they require more resources to train and run than attention-based models. See this post for more info.

But why?

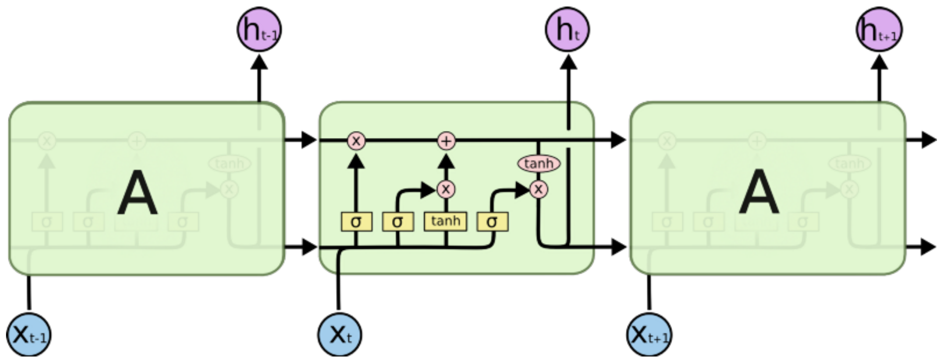
Remember RNN and LSTM and derivatives use mainly sequential processing over time. See the horizontal arrow in the diagram below:



Sequential processing in RNN, from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

This arrow means that long-term information has to sequentially travel through all cells before getting to the present processing cell. This means it can be easily corrupted by being multiplied many time by small numbers < 0. This is the cause of vanishing gradients.

To the rescue, came the LSTM module, which today can be seen as multiple switch gates, and a bit like ResNet it can bypass units and thus remember for longer time steps. LSTM thus have a way to remove some of the vanishing gradients problems.



Sequential processing in LSTM, from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

But not all of it, as you can see from the figure above. Still we have a sequential path from older past cells to the current one. In fact the path is now even more complicated, because it has additive and forget branches attached to it. No question LSTM and GRU and derivatives are able to learn a lot of longer term information! See results here; but they can remember sequences of 100s, not 1000s or 10,000s or more.

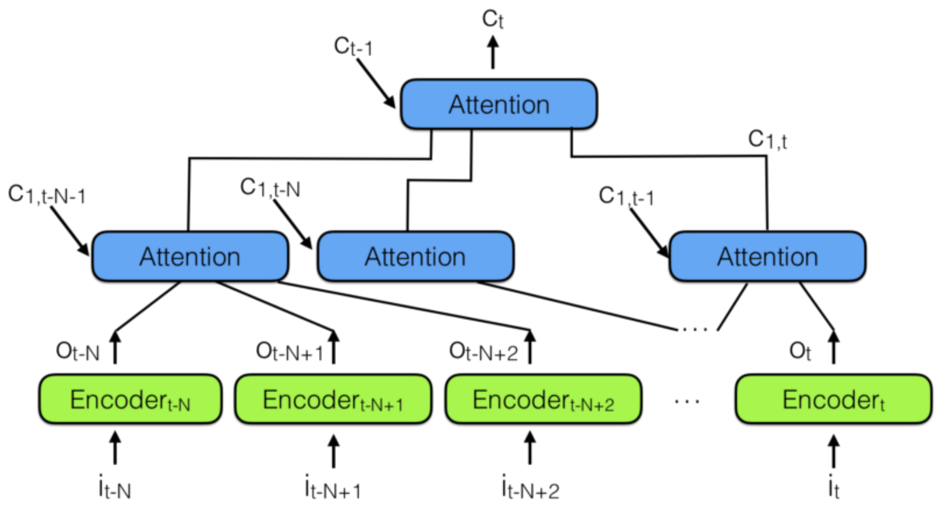
And one issue of RNN is that they are not hardware friendly. Let me explain: it takes a lot of resources we do not have to train these network fast. Also it takes much resources to run these model in the cloud, and given that the demand for speech-to-text is growing rapidly, the cloud is not scalable. We will need to process at the edge, right into the Amazon Echo! See note below for more details.

What do you do?

If sequential processing is to be avoided, then we can find units that "look-ahead" or better "look-back", since most of the time we deal with real-time causal data where we know the past and want to affect future decisions. Not so in translating sentences, or analyzing recorded videos, for example, where we have all data and can reason on it more time.

Such look-back/ahead units are neural attention modules, which we previously explained here.

To the rescue, and combining multiple neural attention modules, comes the “*hierarchical neural attention encoder*”, shown in the figure below:



Hierarchical neural attention encoder

A better way to look into the past is to use attention modules to summarize all past encoded vectors into a context vector Ct.

Notice there is a hierarchy of attention modules here, very similar to the hierarchy of neural networks. This is also similar to Temporal convolutional network (TCN), reported in Note 3 below.

In the *hierarchical neural attention encoder* multiple layers of attention can look at a small portion of recent past, say 100 vectors, while layers above can look at 100 of these attention modules, effectively integrating the information of 100 x 100 vectors. This extends the ability of the *hierarchical neural attention encoder* to 10,000 past vectors.

This is the way to look back more into the past and be able to influence the future.

But more importantly look at the length of the path needed to propagate a representation vector to the output of the network: in hierarchical networks it is proportional to *log(N)* where N are the number of hierarchy layers. This is in contrast to the T steps that a RNN needs to do, where T is the maximum length of the sequence to be remembered, and T >> N.

> *It is easier to remember sequences if you hop 3–4 times, as opposed to hopping 100 times!*

This architecture is similar to a neural Turing machine, but lets the neural network decide what is read out from memory via attention. This means an actual neural network will decide which vectors from the past are important for future decisions.

But what about storing to memory? The architecture above stores all previous representation in memory, unlike neural Turning machines. This can be rather inefficient: think about storing the representation of every frame in a video—most times the representation vector does not change frame-to-frame, so we really are storing too much of the same! What can we do is add another unit to prevent correlated data to be stored. For example by not storing vectors too similar to previously stored ones. But this is really a hack, the best would be to be let the application guide what vectors should be saved or not. This is the focus of current research studies. Stay tuned for more information.

> *So in summary forget RNN and variants. Use attention. Attention really is all you need!*

Tell your friends! It is very surprising to us to see so many companies still use RNN/LSTM for speech to text, many unaware that these networks are so inefficient and not scalable. Please tell them about this post.

Additional information

About training RNN/LSTM: RNN and LSTM are difficult to train because they require memory-bandwidth-bound computation, which is the worst nightmare for hardware designer and ultimately limits the applicability of neural networks solutions. In short, LSTM require 4 linear layer (MLP layer) per cell to run at and for each sequence time-step. Linear layers require large amounts of memory bandwidth to be computed, in fact they cannot use many compute unit often because the system has not enough memory bandwidth to feed the computational units. And it is easy to add more computational units, but hard to add more memory bandwidth (note enough lines on a chip, long wires from processors to memory, etc). As a result, RNN/LSTM and variants are not a good match for hardware acceleration, and we talked about this issue before here and here. A solution will be compute in memory-devices like the ones we work on at FWDNXT.

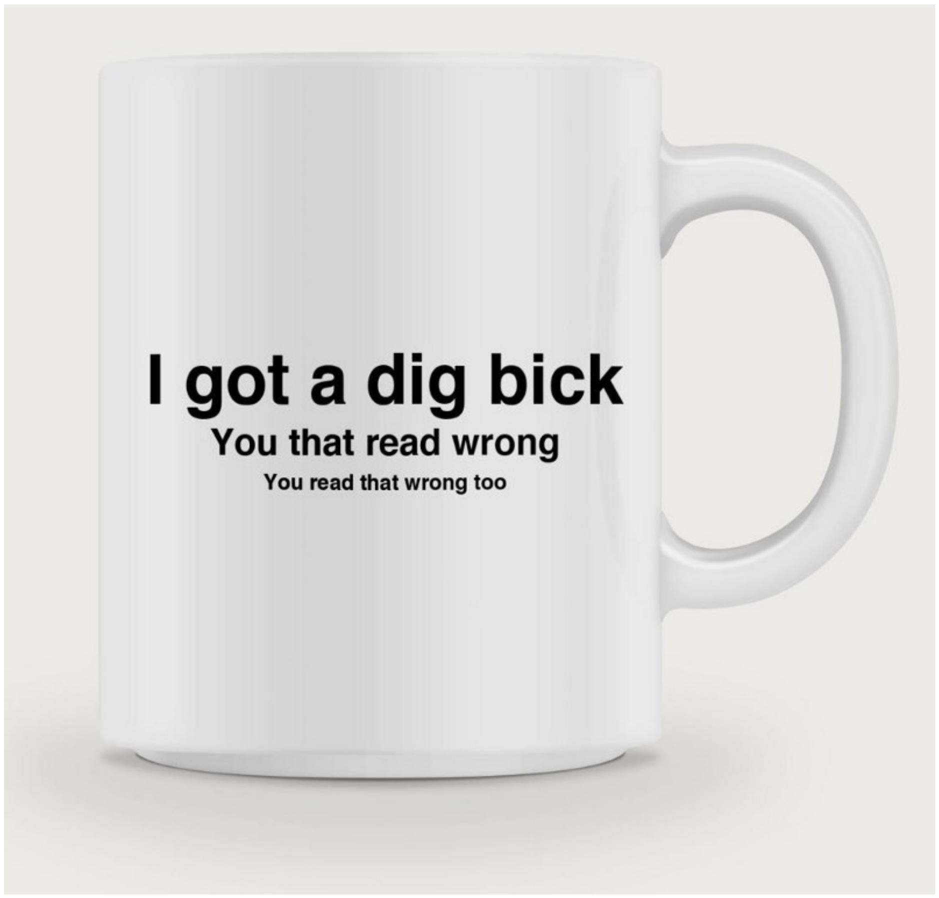
Notes

Note 1: *Hierarchical neural attention* is similar to the ideas in WaveNet. But instead of a convolutional neural network we use hierarchical attention modules. Also: *Hierarchical neural attention* can be also bi-directional.

Note 2: RNN and LSTM are memory-bandwidth limited problems (see this for details). The processing unit(s) need as much memory bandwidth as the number of operations/s they can provide, making it impossible to fully utilize them! The external bandwidth is never going to be enough, and a way to slightly ameliorate the problem is to use internal fast caches with high bandwidth. The best way is to use techniques that do not require large amount of parameters to be moved back and forth from memory, or that can be re-used for multiple computation per byte transferred (high arithmetic intensity).

Note 3: Here is a paper comparing CNN to RNN. Temporal convolutional network (TCN) "outperform canonical recurrent networks such as LSTMs across a diverse range of tasks and datasets, while demonstrating longer effective memory".

Note 4: Related to this topic, is the fact that we know little of how our human brain learns and remembers sequences. "We often learn and recall long sequences in smaller segments, such as a phone number 858 534 22 30 memorized as four segments. Behavioral experiments suggest that humans and some animals employ this strategy of breaking down cognitive or behavioral sequences into chunks in a wide variety of tasks" —these chunks remind me of small convolutional or attention like networks on smaller sequences, that then are hierarchically strung together like in the *hierarchical neural attention encoder* and Temporal convolutional network (TCN). More studies make me think that working memory is similar to RNN networks that uses recurrent real neuron networks, and their capacity is very low. On the other hand both the cortex and hippocampus give us the ability to remember really long sequences of steps (like: where did I park my car at airport 5 days ago), suggesting that more parallel pathways may be involved to recall long sequences, where attention mechanism gate important chunks and force hops in parts of the sequence that is not relevant to the final goal or task.



Note 5: The above evidence shows we do not read sequentially, in fact we interpret characters, words and sentences as a group. An attention-based or convolutional module perceives the sequence and projects a representation in our mind. We would not be misreading this if we processed this information sequentially! We would stop and notice the inconsistencies!

About the author

I have almost 20 years of experience in neural networks in both hardware and software (a rare combination). See about me here: Medium, webpage, Scholar, LinkedIn, and more…

Donations

If you found this article useful, please consider a donation to support more tutorials and blogs. Any contribution can make a difference!