

Understanding LSTM Networks

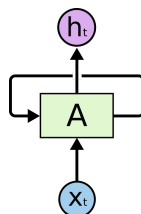
Posted on August 27, 2015

Recurrent Neural Networks

Humans don't start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. You don't throw everything away and start thinking from scratch again. Your thoughts have persistence.

Traditional neural networks can't do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

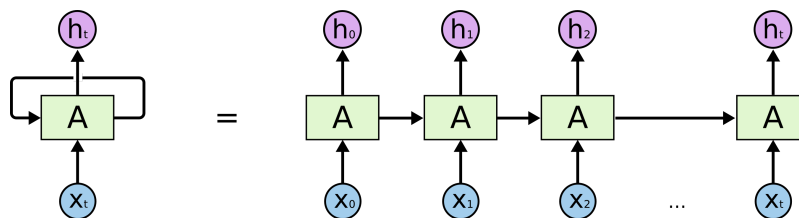
Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.



Recurrent Neural Networks have loops.

In the above diagram, a chunk of neural network, A , looks at some input x_t and outputs a value h_t . A loop allows information to be passed from one step of the network to the next.

These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren't all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:



An unrolled recurrent neural network.

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data.

And they certainly are used! In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning... The list goes on. I'll leave discussion of the amazing feats one can achieve with RNNs to Andrej Karpathy's excellent blog post, [The Unreasonable Effectiveness of](#)

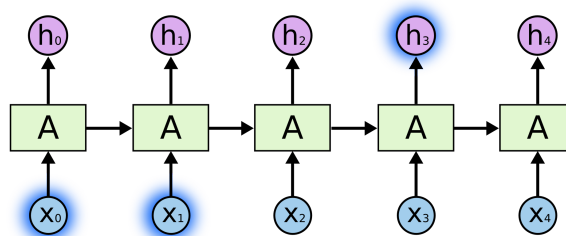
Recurrent Neural Networks (<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>). But they really are pretty amazing.

Essential to these successes is the use of “LSTMs,” a very special kind of recurrent neural network which works, for many tasks, much much better than the standard version. Almost all exciting results based on recurrent neural networks are achieved with them. It’s these LSTMs that this essay will explore.

The Problem of Long-Term Dependencies

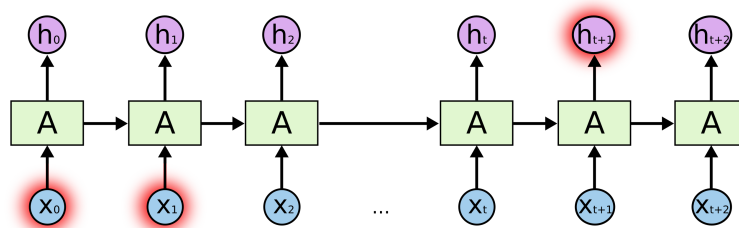
One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame. If RNNs could do this, they’d be extremely useful. But can they? It depends.

Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in “the clouds are in the *sky*,” we don’t need any further context – it’s pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it’s needed is small, RNNs can learn to use the past information.



But there are also cases where we need more context. Consider trying to predict the last word in the text “I grew up in France... I speak fluent *French*.” Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large.

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.



In theory, RNNs are absolutely capable of handling such “long-term dependencies.” A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don’t seem to be able to learn them. The problem was explored in depth by Hochreiter (1991) [German] (<http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf>) and Bengio, et al. (1994) (<http://www-dsi.ing.unifi.it/~paolo/ps/tnn-94-gradient.pdf>), who found some pretty fundamental reasons why it might be difficult.

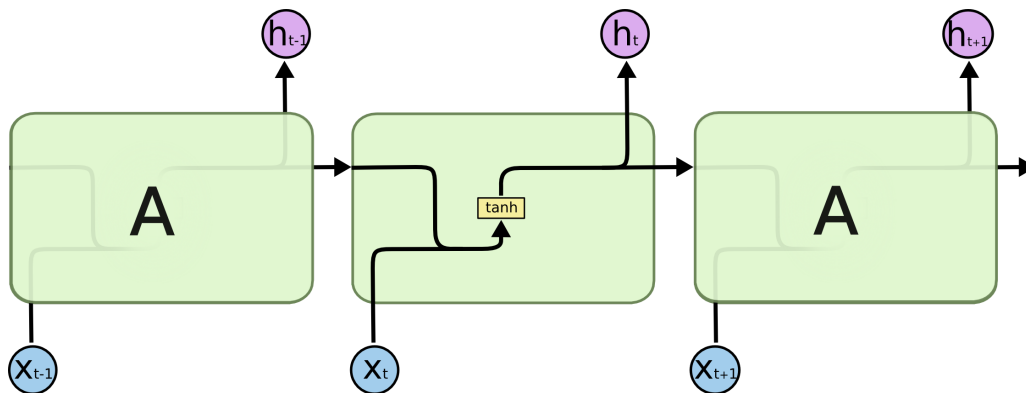
Thankfully, LSTMs don’t have this problem!

LSTM Networks

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997) (http://deeplearning.cs.cmu.edu/pdfs/Hochreiter97_lstm.pdf), and were refined and popularized by many people in following work.¹ They work tremendously well on a large variety of problems, and are now widely used.

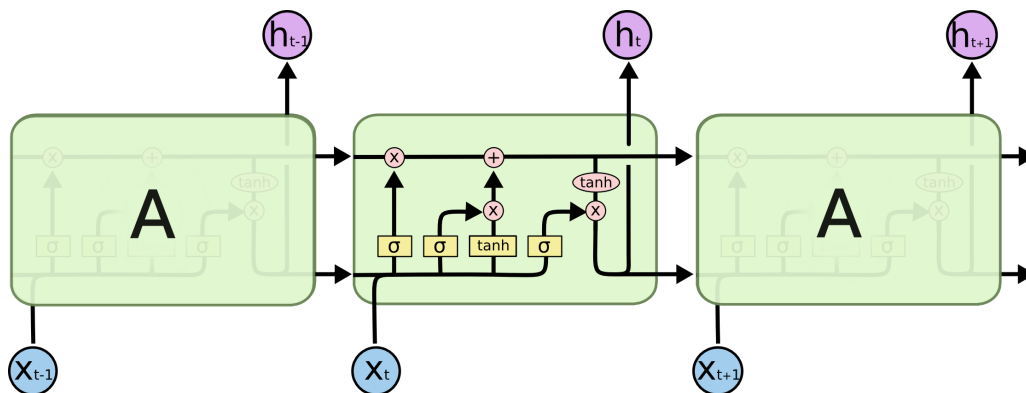
LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



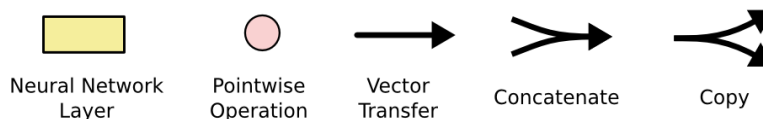
The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.

Don't worry about the details of what's going on. We'll walk through the LSTM diagram step by step later. For now, let's just try to get comfortable with the notation we'll be using.

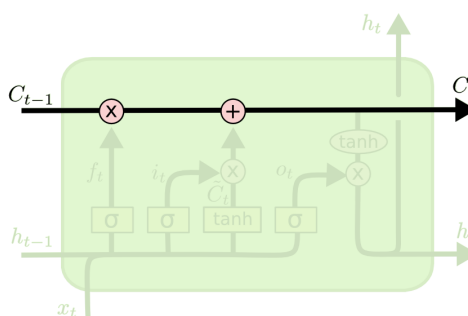


In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denotes its content being copied and the copies going to different locations.

The Core Idea Behind LSTMs

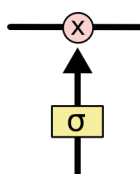
The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



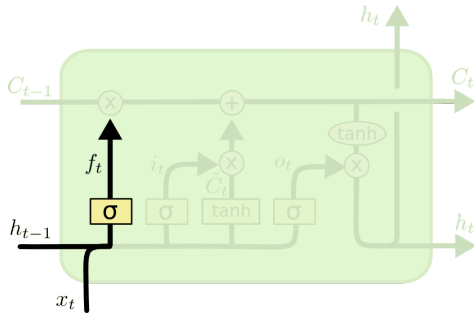
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”

An LSTM has three of these gates, to protect and control the cell state.

Step-by-Step LSTM Walk Through

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the “forget gate layer.” It looks at h_{t-1} and x_t , and outputs a number between 0 and 1 for each number in the cell state C_{t-1} . A 1 represents “completely keep this” while a 0 represents “completely get rid of this.”

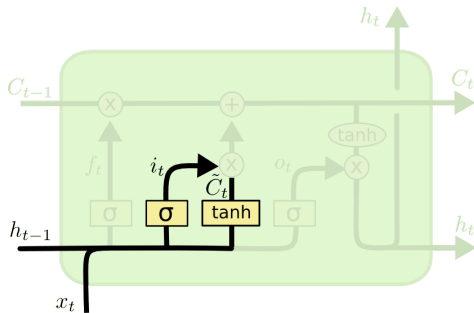
Let's go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, \tilde{C}_t , that could be added to the state. In the next step, we'll combine these two to create an update to the state.

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.



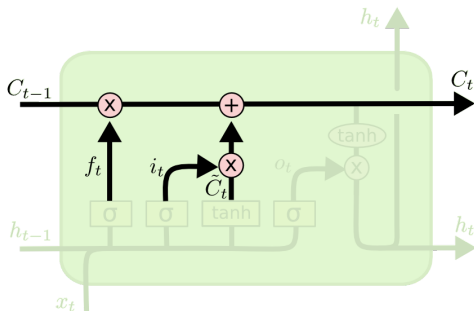
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

It's now time to update the old cell state, C_{t-1} , into the new cell state C_t . The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by f_t , forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

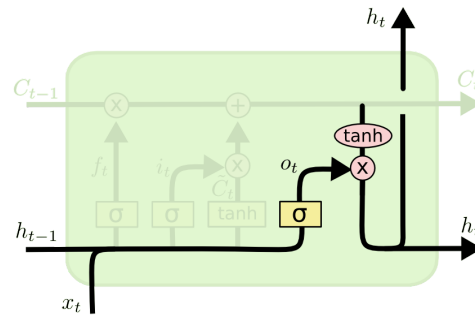
In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through \tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.



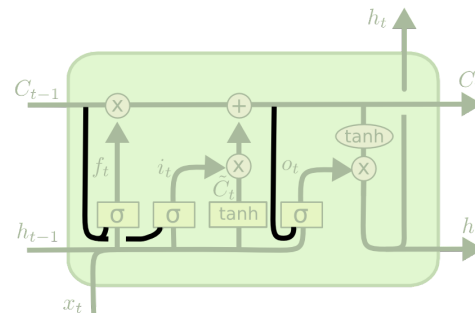
$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Variants on Long Short Term Memory

What I've described so far is a pretty normal LSTM. But not all LSTMs are the same as the above. In fact, it seems like almost every paper involving LSTMs uses a slightly different version. The differences are minor, but it's worth mentioning some of them.

One popular LSTM variant, introduced by Gers & Schmidhuber (2000) (<ftp://ftp.idsia.ch/pub/juergen/TimeCount-LJCNN2000.pdf>), is adding "peephole connections." This means that we let the gate layers look at the cell state.



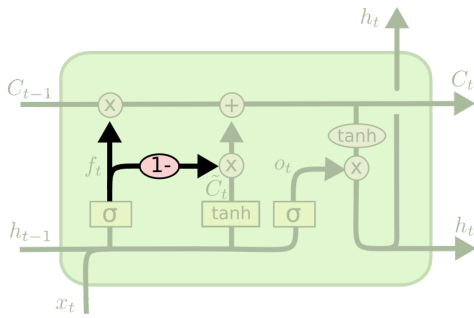
$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$

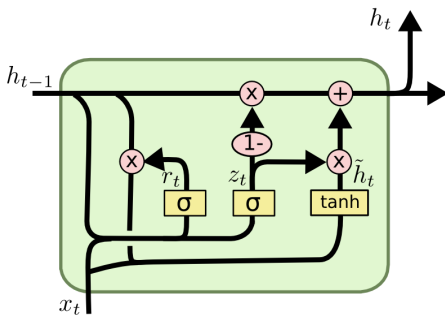
The above diagram adds peepholes to all the gates, but many papers will give some peepholes and not others.

Another variation is to use coupled forget and input gates. Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.



$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014) (<http://arxiv.org/pdf/1406.1078v3.pdf>). It combines the forget and input gates into a single “update gate.” It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

These are only a few of the most notable LSTM variants. There are lots of others, like Depth Gated RNNs by Yao, et al. (2015) (<http://arxiv.org/pdf/1508.03790v2.pdf>). There's also some completely different approach to tackling long-term dependencies, like Clockwork RNNs by Koutnik, et al. (2014) (<http://arxiv.org/pdf/1402.3511v1.pdf>).

Which of these variants is best? Do the differences matter? Greff, et al. (2015) (<http://arxiv.org/pdf/1503.04069.pdf>) do a nice comparison of popular variants, finding that they're all about the same. Jozefowicz, et al. (2015) (<http://jmlr.org/proceedings/papers/v37/jozefowicz15.pdf>) tested more than ten thousand RNN architectures, finding some that worked better than LSTMs on certain tasks.

Conclusion

Earlier, I mentioned the remarkable results people are achieving with RNNs. Essentially all of these are achieved using LSTMs. They really work a lot better for most tasks!

Written down as a set of equations, LSTMs look pretty intimidating. Hopefully, walking through them step by step in this essay has made them a bit more approachable.

LSTMs were a big step in what we can accomplish with RNNs. It's natural to wonder: is there another big step? A common opinion among researchers is: “Yes! There is a next step and it's attention!” The idea is to let every step of an RNN pick information to look at from some larger collection of information. For example, if you are using an RNN to create a caption describing an image, it might pick a part of the image to look at for every word it outputs. In fact, Xu, *et al.* (2015) (<http://arxiv.org/pdf/1502.03044v2.pdf>) do exactly this – it might be a fun starting point if you want to explore attention! There's been a number of really exciting results using attention, and it seems like a lot more are around the corner...

Attention isn't the only exciting thread in RNN research. For example, Grid LSTMs by Kalchbrenner, *et al.* (2015) (<http://arxiv.org/pdf/1507.01526v1.pdf>) seem extremely promising. Work using RNNs in generative models – such as Gregor, *et al.* (2015) (<http://arxiv.org/pdf/1502.04623.pdf>), Chung, *et al.* (2015) (<http://arxiv.org/pdf/1506.02216v3.pdf>), or Bayer & Osendorfer (2015) (<http://arxiv.org/pdf/1411.7610v3.pdf>) – also seems very interesting. The last few years have been an exciting time for recurrent neural networks, and the coming ones promise to only be more so!

Acknowledgments

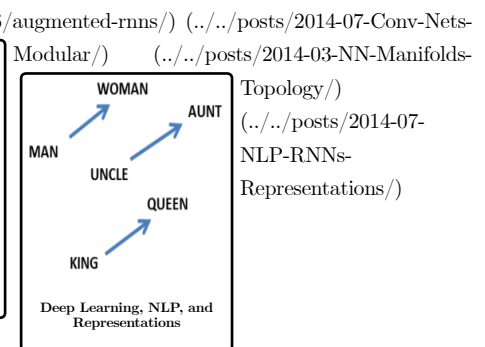
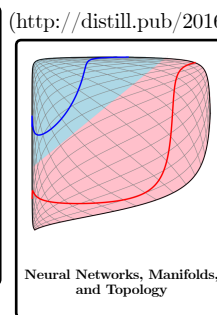
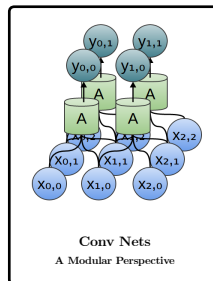
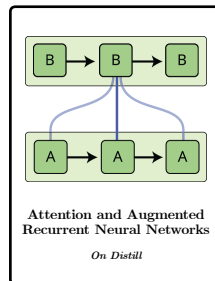
I'm grateful to a number of people for helping me better understand LSTMs, commenting on the visualizations, and providing feedback on this post.

I'm very grateful to my colleagues at Google for their helpful feedback, especially Oriol Vinyals (<http://research.google.com/pubs/OriolVinyals.html>), Greg Corrado (<http://research.google.com/pubs/GregCorrado.html>), Jon Shlens (<http://research.google.com/pubs/JonathonShlens.html>), Luke Vilnis (<http://people.cs.umass.edu/~luke/>), and Ilya Sutskever (<http://www.cs.toronto.edu/~ilya/>). I'm also thankful to many other friends and colleagues for taking the time to help me, including Dario Amodei (<https://www.linkedin.com/pub/dario-amodei/4/493/393>), and Jacob Steinhardt (<http://cs.stanford.edu/~jsteinhardt/>). I'm especially thankful to Kyunghyun Cho (<http://www.kyunghyuncho.me/>) for extremely thoughtful correspondence about my diagrams.

Before this post, I practiced explaining LSTMs during two seminar series I taught on neural networks. Thanks to everyone who participated in those for their patience with me, and for their feedback.

1. In addition to the original authors, a lot of people contributed to the modern LSTM. A non-comprehensive list is: Felix Gers, Fred Cummins, Santiago Fernandez, Justin Bayer, Daan Wierstra, Julian Togelius, Faustian Gomez, Matteo Gagliolo, and Alex Graves (<https://scholar.google.com/citations?user=DaFHynwAAAAJ&hl=en>).↩

More Posts



174 Comments (/posts/2015-08-Understanding-LSTMs/#disqus_thread)

174 Comments colah's blog

Login

Recommend 90 Share

Sort by Best



Join the discussion...

**Dan Quang** • a year ago

Thank you! I've been trying to understand LSTM for a long time! This made it very clear :)

12 ^ | v • Reply • Share ›

**chrisolah** Mod → Dan Quang • a year ago

I'm glad it helped!

^ | v • Reply • Share ›

**Dan Quang** → chrisolah • a year ago

I have to give a presentation soon to my department explaining LSTMs for my candidacy exam. Do you mind if I use some of these figures to explain LSTMs?

^ | v • Reply • Share ›

**chrisolah** Mod → Dan Quang • 10 months ago

Sorry I didn't respond earlier. For you, and anyone else reading, I'm happy for anyone to use these figures with attribution.

6 ^ | v • Reply • Share ›

**Brandon Rohrer** • 10 months ago

Thank you for the clear explanation! It's a rare that someone both has the ability to write so clearly and takes the time to do so. Your work is a gift to the rest of us.

9 ^ | v • Reply • Share ›

**li kai** • a year ago

Great!Great!Great!Great!Great!Great!Great!Great!Great!

Much better than Alex Graves's paper!!!!

9 ^ | v • Reply • Share ›

**Daniel Bigham** → li kai • a year ago

LOL. I was just reading Alex Grave's paper and started to frown with the lack of explanation on the core LSTM idea. I googled, found this article, and your comment was at the top. Hilarious.

3 ^ | v • Reply • Share ›

**Alex Sosnovshchenko** • a year ago

I made Russian translation of this excellent post, I hope you will not mind.

<http://alexsosn.github.io/ml/2...>

5 ^ | v • Reply • Share ›

**bobriakov** → Alex Sosnovshchenko • 9 months ago

Спасибо!

1 ^ | v • Reply • Share ›

**chrisolah** Mod → Alex Sosnovshchenko • a year ago

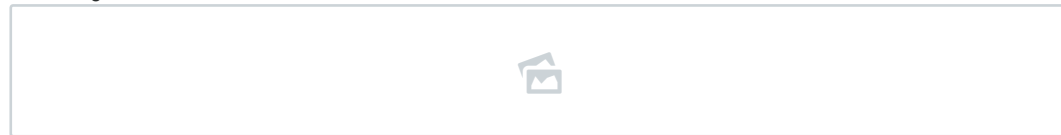
That looks lovely, Alex! Thanks for the translation!

^ | v • Reply • Share ›

**Steven Tang** • a year agoGreat post! I am a little confused by the output h_t portion of the lstm.It looks like the dimension of C_t should be the number of dimensions in h_{t-1} and x_t combined.It also looks like h_t should be the dimension of C_t .Obviously, my understanding can't be correct, or else the dimensionality of h_t would grow by the dimensions of x_t at

every time step. Is there a mistake in my understanding somewhere? Is the output of h_t only the dimension of h_{t-1} , and NOT C_t ?

Thanks again!



2 ^ | v • Reply • Share ›



Atcold → Steven Tang • a year ago

— | 🚩

h_t , C_t and \tilde{C}_t share the same dimensionality.

The size of \tilde{C}_t comes directly from the 'height' of W_C matrix (check \tilde{C}_t definition, above). W_C 's 'width' is instead equal to $\text{size}(h_t) + \text{size}(x_t)$.

2 ^ | v • Reply • Share ›



Majid al-Dosari → Atcold • a year ago

— | 🚩

i don't get it. everything that's not a weight matrix is a vector of the same dimension. i don't get what you mean by this concatenation thing mathematically $W_C[h_{(t-1)}, x_t]$. it seems unconventional.

do you mean 'wide' matrix times 'tall' vector b/c you put h and x on top of each other?? that would equal a tall vector which can't be right. it can't be two vectors side by side b/c then the result would be a matrix.

in the literature the weight matrix is separated out for each vector.

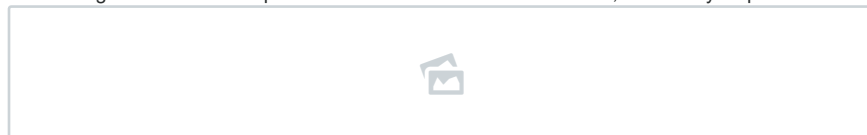
^ | v • Reply • Share ›



chrisolah Mod → Majid al-Dosari • 10 months ago

— | 🚩

Here's a figure I made for the post but didn't include in the final version, which may help:



3 ^ | v • Reply • Share ›



hughperkins → chrisolah • 10 months ago

— | 🚩

These diagrams look really nice. What are you using to draw them?

6 ^ | v • Reply • Share ›



Carl Thomé → hughperkins • 19 days ago

— | 🚩

Chris mentioned he did them in Inkscape. Beautiful stuff, indeed!

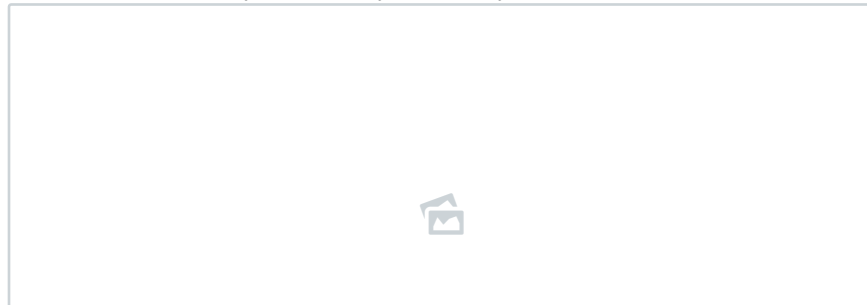
1 ^ | v • Reply • Share ›



Majid al-Dosari → chrisolah • 10 months ago

— | 🚩

thx. i made a figure similar to yours actually (might need some more arrows) it's funny how in the literature the focus is usually on a 'cell' but you can't really see the data flow.



^ | v • Reply • Share ›



chrisolah Mod → Majid al-Dosari • 10 months ago

Nice! :)

^ | v • Reply • Share ›



Ibrahim Sobh → Majid al-Dosari • 5 months ago

please have a look at:
<http://jmlr.org/proceedings/pa...>

^ | v • Reply • Share ›



Majid al-Dosari → Majid al-Dosari • a year ago

oh i get it... two vectors on top of each other and height of W is size of vector.

^ | v • Reply • Share ›



Atcold → Majid al-Dosari • a year ago

Exactly. $Ax + By = [A|B][x/y]$, where $|$ concatenates horizontally and $/$ vertically.

1 ^ | v • Reply • Share ›



Steven Tang → Atcold • a year ago

Great, thanks for the explanation!

^ | v • Reply • Share ›



Steven Yin • a year ago

Great post! Seems like RNN is great at predicting stuff. I am trying to see if RNN can be applied to time series classification. Do you have any papers you would suggest relating to that? Or are there any other types of Neural nets that are good at time series classification?

2 ^ | v • Reply • Share ›



Carl Thomé → Steven Yin • 19 days ago

That's one of the most common applications for RNNs. Here's a good starting point in Keras:
<https://github.com/fchollet/ke...>

^ | v • Reply • Share ›



Bryan Baek • 3 months ago

Really nice explanation about the LSTM. I was confused a lot but I just got quite better understanding. Thanks a lot!

1 ^ | v • Reply • Share ›



Beheen Trimble • 3 months ago

Thank you.

1 ^ | v • Reply • Share ›



Weijie Huang • 9 months ago

In fact, how to calculate the " $Wf[H_{t-1}, X_t]$ "? Is it the same as $(Wf_1 * H_{t-1} + Wf_2 * X_t)$?

1 ^ | v • Reply • Share ›



gaurav gupta → Weijie Huang • 2 months ago

I think they are same. h_{t-1} is stacked on the top of x_t to form a single vertical one D vector. Let the total height be N. So Wf is a matrix of height N and width M. So $Wf[h_{t-1}/x_t] ==> [M \times N][N \times 1]$ gives $[M \times 1]$. this the dimension of C_t .

^ | v • Reply • Share ›



Ibrahim Sobh → Weijie Huang • 5 months ago

I have the same question, do you have an answer?

please have a look at:

<http://jmlr.org/proceedings/pa...>

^ | v • Reply • Share ›



Ghislain Gagne • 10 months ago

Excellent post, it made me understand better the inner workings of LSTM networks. Are LSTM nets suitable for time series prediction?

thanks

1 ^ | v • Reply • Share ›



Rafael Espericueta → Ghislain Gagne • 4 months ago

That's what LSTMs are for.

^ | v • Reply • Share ›



David Krueger • a year ago

Anyone's thoughts on IRNNs?

<http://arxiv.org/pdf/1504.0094...>

1 ^ | v • Reply • Share ›



Anshul Mittal → David Krueger • a year ago

Can you specify exactly what you wanna ask?

^ | v • Reply • Share ›



Jerry → Anshul Mittal • a year ago

anshul mittal greatest FAC of all time

^ | v • Reply • Share ›



Zachary Jablons • a year ago

Excellent post!

I found the GRU diagram a bit hard to follow, I think it could be improved a bit by adding the equation labels.

1 ^ | v • Reply • Share ›



chrisolah Mod → Zachary Jablons • a year ago

I've added the labels -- thanks for pointing that out!

^ | v • Reply • Share ›



Jan Alberts • 2 days ago



[see more](#)[^](#) | [v](#) | [Reply](#) | [Share](#) >**Elisa Mussumeci** • 5 days ago

Thank you so much for this article, it helped me a lot to learn LSTMs!

[^](#) | [v](#) | [Reply](#) | [Share](#) >**Faiyaz Lalani** • 7 days ago

Nice post Chris - just wanted to thank you.

[^](#) | [v](#) | [Reply](#) | [Share](#) >**Chris Anderson** • 10 days ago

I think your equations for the GRU may be wrong. Looking at the original GRU paper (this one I think: <https://arxiv.org/pdf/1406.107...> on page 3), you've got the final activation wrong - got your (z) and (1-z) backwards. There are other issues too - might be worth a check.

[^](#) | [v](#) | [Reply](#) | [Share](#) >**Raju Krishnan** • 14 days ago

Hi Chris - thanks for this post. How is an LSTM network trained? Do the usual gradient descent first and second order methods work?

[^](#) | [v](#) | [Reply](#) | [Share](#) >**Carl Thomé** → **Raju Krishnan** • 14 days ago

Yes, they are trained normally by taking steps in the gradient's direction. However, in order to know the gradients for weights before the non-linear activation functions additional steps are required, specifically backpropagation through time (BPTT): <https://colah.github.io/posts/...>

[^](#) | [v](#) | [Reply](#) | [Share](#) >**Abhishek Mishra** • 20 days ago

I was wondering the reason behind calling LSTM units Long Short Term Memories. Please let me know your opinion.

[^](#) | [v](#) | [Reply](#) | [Share](#) >**Florian Courtial** → **Abhishek Mishra** • 19 days ago

Because recurrent neural networks have a memory, LSTM are a kind of RNN that offer a longer memory (because of their inner mechanic).

[^](#) | [v](#) | [Reply](#) | [Share](#) >**Abhishek Mishra** → **Florian Courtial** • 19 days ago

Is it correct to assume that "Long Memory" stands for "Cell State" and "Short Memory" stands for "Hidden State"?


[^](#) | [v](#) | [Reply](#) | [Share](#) >**Carl Thomé** → **Abhishek Mishra** • 19 days ago

No. The name "long short-term memory" is just a play-on-words, and LSTMs have nothing at all to do with long-term memory. They only mimic short-term memory, but for long sequences (hence the joke).


[^](#) | [v](#) | [Reply](#) | [Share](#) >**Zhipeng Wang** • a month ago

I am new to Deep Learning so this post is really helpful. Just out of curiosity, are those deep learning architectures come


out of intuition? Do they have solid theory behind it to prove that they are good for certain tasks?
^ | v • Reply • Share ›



Carl Thomé ➔ Zhipeng Wang • 19 days ago
Mostly intuition, I think. Some of it was sprung out of ideas of how to emulate human learning (see Hebbian learning) but then rethought to actually work (e.g. backprop). People are constantly coming up with solid theory for why stuff works (Yarin Gal, comes to mind), but a lot of the intuition is still only that. Stack more layers!
^ | v • Reply • Share ›



Md. Al-Amin • 2 months ago
Thanks. Very nice article, helped me a lot :)
^ | v • Reply • Share ›



Caleb Rascon • 2 months ago
This was a great post. It really helped me get my head around the concept behind LTSM-RNNs. Thanks a lot for this.
^ | v • Reply • Share ›

Load more comments

ALSO ON COLAH'S BLOG

Conv Nets: A Modular Perspective
2 comments • a year ago•
Atcold — It's a logistic sigmoid function. Or, any non linearity, more in general.

Groups & Group Convolutions
1 comment • 17 days ago•
Ryan Gnabasik — You point out something that I think is true for vector spaces as well. In general vector spaces inherit properties from the fields ...

Visual Information Theory
27 comments • a year ago•
Heng, Wei — Great article! Just be curious about what tool you use to plot these figures?

Calculus on Computational Graphs: Backpropagation
1 comment • a year ago•
Jedd Haberstro — Thank you for the insightful article. If it were possible not to do so in an overly complicating way, the expression tree example ...

