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.
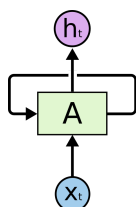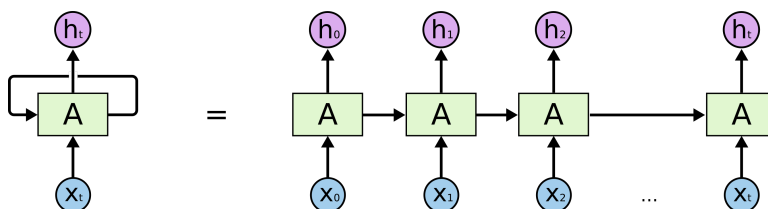

Recurrent Neural Networks have loops.

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.
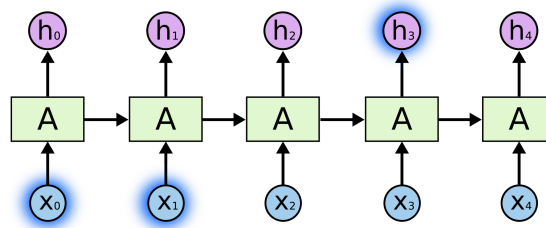
But there are also cases where we need more context. Consider trying to predict the last word in the text "I grew up in France... I speak fluent *French*." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context 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.
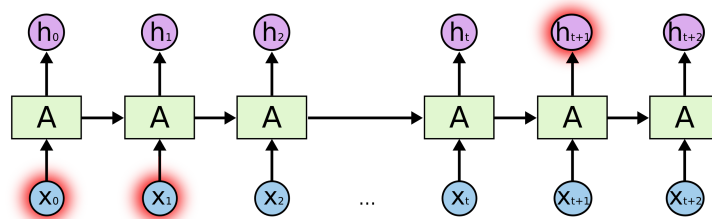
In theory, RNNs are absolutely capable of handling such "long-term dependencies." A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don't seem to be able to learn them. The problem was explored in depth by Hochreiter (1991) [German] (http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.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://www.bioinf.jku.at/publications/older/2604.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.
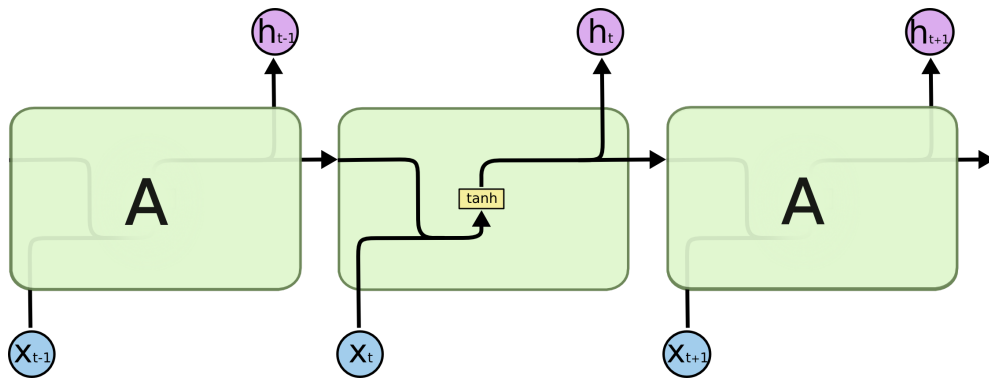

The repeating module in a standard RNN contains a single layer.

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.
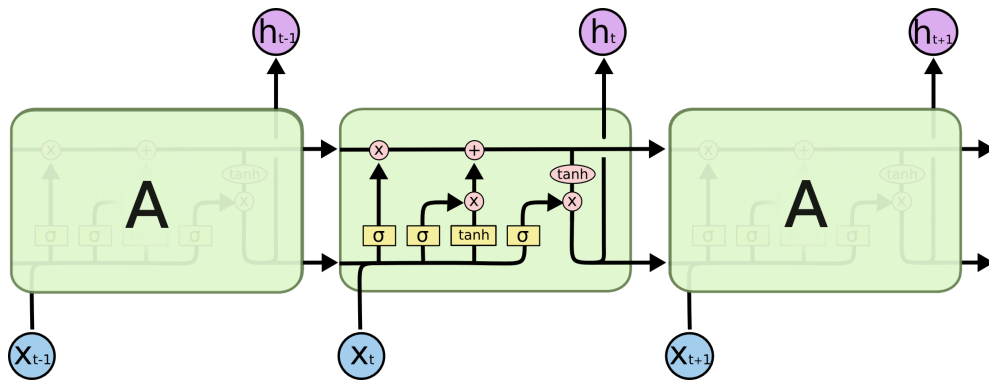
The repeating module in an LSTM contains four interacting layers.

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.
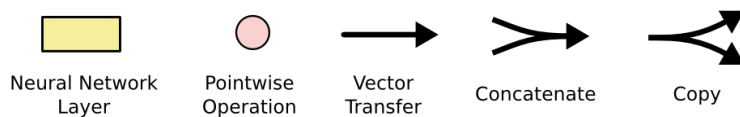

In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote 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.
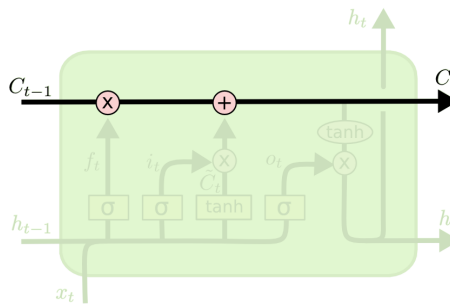
The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

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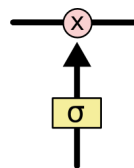

The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

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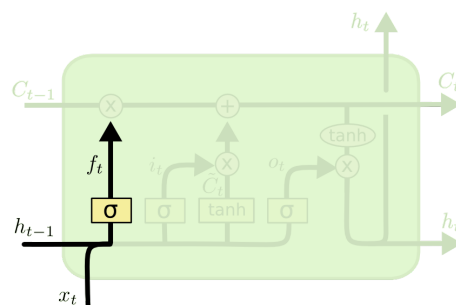
$$f_t = \sigma \left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f \right)$$

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.
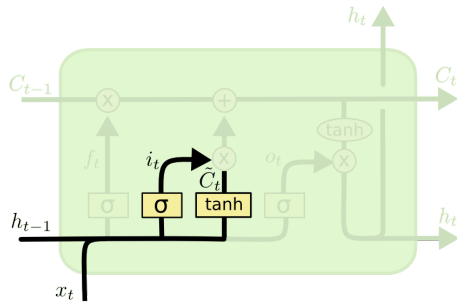
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

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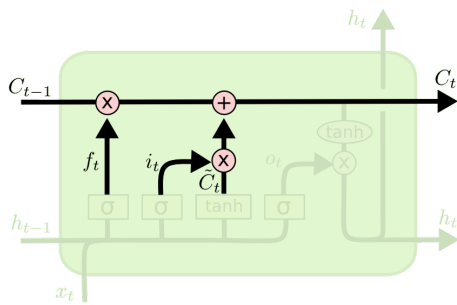
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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.
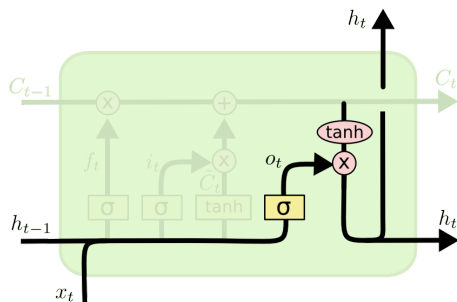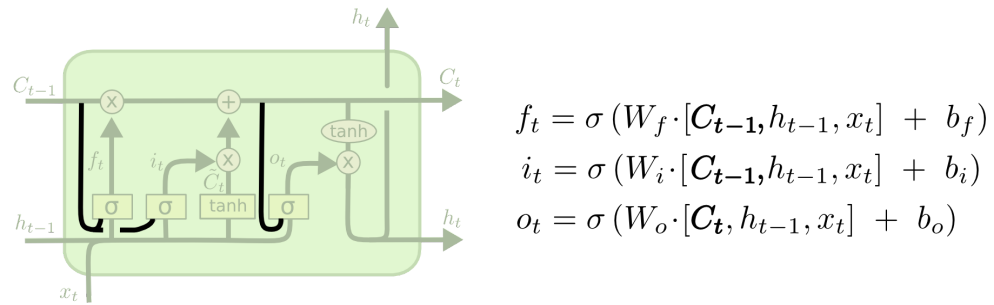
$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

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-IJCNN2000.pdf), is adding "peephole connections." This means that we let the gate layers look at the cell state.

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_i\right)$$
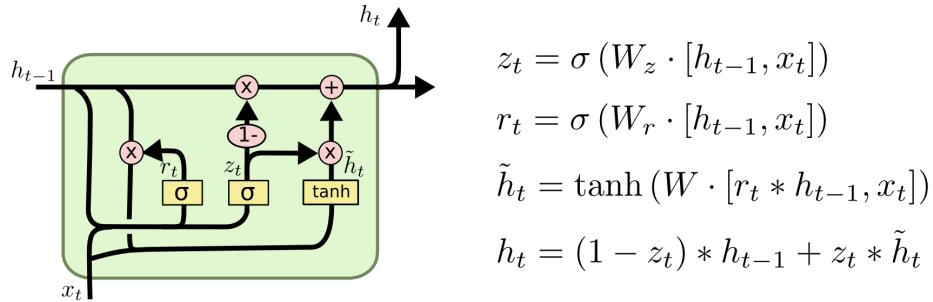$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] + b_o\right)$$

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.

$$C_t = f_t * C_{t-1} + (\boldsymbol{1 - f_t}) * \tilde{C}_t$$
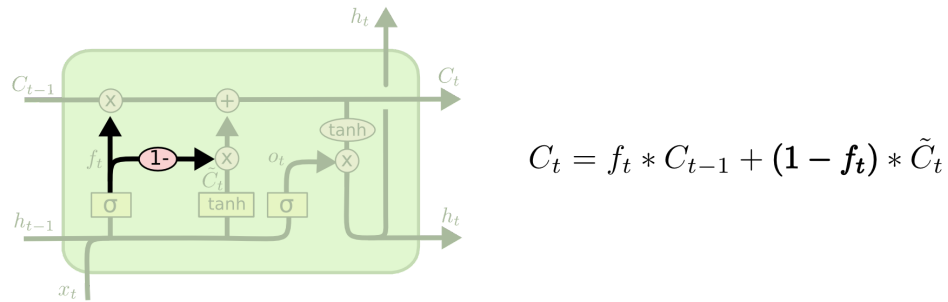
A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014) (http://arxiv.org/pdf/1406.1078v3.pdf). It combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

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!
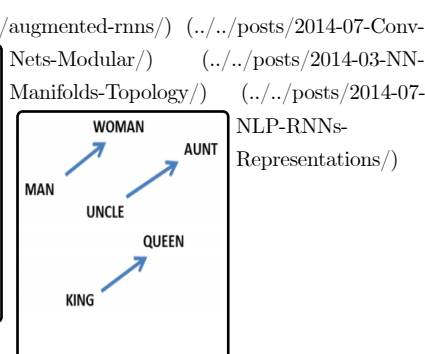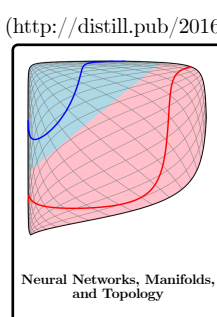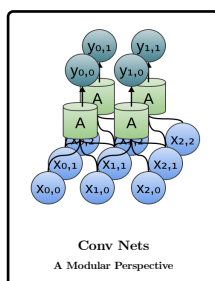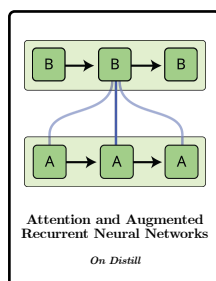
Acknowledgments

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

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, Faustino Gomez, Matteo Gagliolo, and Alex Graves (https://scholar.google.com/citations?user=DaFHynwAAAAJ&hl=en).↵

More Posts

Attention and Augmented Recurrent Neural Networks

On Distill

Conv Nets
A Modular Perspective

(http://distill.pub/2016/augmented-rnns/) (../../posts/2014-07-Conv-Nets-Modular/) (../../posts/2014-03-NN-Manifolds-Topology/) (../../posts/2014-07-NLP-RNNs-Representations/)

Neural Networks, Manifolds, and Topology

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

347 Comments   **colah's blog**   ● 1 **Login** ⌄

♡ **Recommend** 283   ⬆ **Share**   Sort by Best ⌄

[avatar]   ┌───┐
                  │ Join the discussion…   │
                  └───┘

                  LOG IN WITH   **OR SIGN UP WITH DISQUS** ?

                                          ┌─────────────────────────────────────┐
                                          │ Name   │
                                          └─────────────────────────────────────┘

Brandon Rohrer • 2 years 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.
157 ⌃ ┊ ⌄ • Reply • Share ›

> **Yogi Tri Cahyono** ➜ Brandon Rohrer • 10 months ago
> true
> 1 ⌃ ┊ ⌄ • Reply • Share ›

> **이인묵** ➜ Brandon Rohrer • 9 months ago
> absolutly
> 1 ⌃ ┊ ⌄ • Reply • Share ›

> **Enish Paneru** ➜ Brandon Rohrer • 8 days ago
> I just finished your video on RNN and LSTM minutes ago.
> Your work is a gift as well. Thank you very much for your effort.
> ⌃ ┊ ⌄ • Reply • Share ›

> **Huy Hoang** ➜ Brandon Rohrer • a month ago
> So true :)
> ⌃ ┊ ⌄ • Reply • Share ›

> **Josue Valdez** ➜ Brandon Rohrer • 3 months ago
> I like your videos
> ⌃ ┊ ⌄ • Reply • Share ›

>> **qinlu zhang** ➜ Josue Valdez • 3 months ago
>> can you post the link of his video
>> ⌃ ┊ ⌄ • Reply • Share ›

>>> **Josue Valdez** ➜ qinlu zhang • 3 months ago
>>> https://disq.us/url?url=htt...
>>> ⌃ ┊ ⌄ • Reply • Share ›

>> **George William** ➜ Josue Valdez • 3 months ago
>> can you post the link of his video
>> ⌃ ┊ ⌄ • Reply • Share ›

>>> **Josue Valdez** ➜ George William • 3 months ago
>>> ┌─────────────────────────────────────┐
>>> │   │
>>> │   │
>>> └─────────────────────────────────────┘

∧ | ∨ • Reply • Share ›

li kai • 3 years ago
Great!Great!Great!Great!Great!Great!Great!Great!Great!Great!

Much better than Alex Graves's paper!!!!
26 ∧ | ∨ • Reply • Share ›

> **Daniel Bigham** ➜ li kai • 3 years 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.
> 15 ∧ | ∨ • Reply • Share ›

Dieka Nugraha • a year ago
Thank you for the most informative and clear post about LSTM in the net with superb visualization!
about the dimensionality of weight matrices, W_f, W_i, W_o have same shape, which is dimension of [h_t-1, x_t] X dimension of Ct right?
24 ∧ | ∨ • Reply • Share ›

> **chrisolah** Mod ➜ Dieka Nugraha • a year ago
> That's right! :)
> 1 ∧ | ∨ • Reply • Share ›

Faiyaz Lalani • 2 years ago
Nice post Chris - just wanted to thank you.
21 ∧ | ∨ • Reply • Share ›

Dan Quang • 3 years ago
Thank you! I've been trying to understand LSTM for a long time! This made it very clear :)
31 ∧ | ∨ • Reply • Share ›

> **chrisolah** Mod ➜ Dan Quang • 3 years ago
> I'm glad it helped!
> 3 ∧ | ∨ • Reply • Share ›

> > **Dan Quang** ➜ chrisolah • 3 years 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?
> > 17 ∧ | ∨ • Reply • Share ›

> > > **chrisolah** Mod ➜ Dan Quang • 2 years ago
> > > Sorry I didn't respond earlier. For you, and anyone else reading, I'm happy for anyone to use these figures with attribution.
> > > 20 ∧ | ∨ • Reply • Share ›

> > > **Morshed Derbali** ➜ Dan Quang • 9 months ago
> > > hello dear ..i'm new in this domain(LSTMs) ..i'm asking you if you permit to pass the presentation to me ..Thanks in advance
> > > morshed.derbali28@gmail.com
> > > ∧ | ∨ • Reply • Share ›

Srinidhi Bheesette ➔ chrisolah • a year ago
Can these be used for particle pattern recognition?
∧ | ∨ • Reply • Share ›

Long Ye • 7 months ago
What is the structure like if there are 128 units in A, no a single unit?
20 ∧ | ∨ • Reply • Share ›

rohan chikorde • a year ago
Thank you for the nice and clear explanation.I really appreciate that. As I am new to this model and learning so just had few doubts:
1) What is b(c), b(i), b(f) in the 1st, 2nd, 3rd equations? what "b" stands for?
2) W(f), W(i), W(o) are neural network matrix but how are they created and why are there 3 separate matrix? Please put some light on this.
3) In one of the equation, we are using tanh and why not sigmoid? Do you use tanh only for range -1 to 1?

Please explain these doubts. It will help us to understand this model more clearly. Thanks in advance.
20 ∧ | ∨ • Reply • Share ›

chrisolah Mod ➔ rohan chikorde • a year ago
1) b_c, b_i, b_f ... are neural network bias vectors. They are initialized with random numbers, and learned as the network trains.
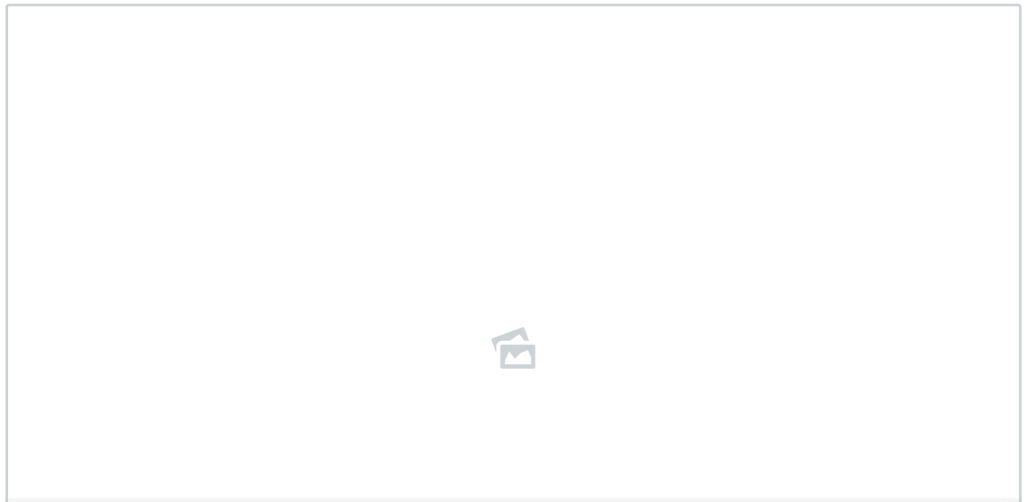
2) W_f, W_i, W_o ... are neural network weight matrices. They are initialized with random numbers, and learned as the network trains.

3) Yes, we use tanh for the range (-1,1)
2 ∧ | ∨ • Reply • Share ›

Jan Alberts • 2 years ago

see more
17 ∧ | ∨ • Reply • Share ›

chrisolah Mod ➔ Jan Alberts • a year ago
Hi Jan -- that's a great question!

In a normal recurrent neural network, ever part of the state is connected to every part of the state.

LSTMs are slightly different. The LSTM cell states don't directly modify themselves. They're carefully protected by design, so that their default behavior is to not change. But every cell does effect the forget gate, input gate, and tanh later. Together, all those layers decide how to change the cell state.

So, all the cell states can change each other, but they do it indirectly, through the special layers we've set

up.

^ | ∨ • Reply • Share ›

Alex Sosnovshchenko • 2 years ago

I made Russian translation of this excellent post, I hope you will not mind.
http://alexsosn.github.io/m...

14 ^ | ∨ • Reply • Share ›

> **bobriakov** ➜ Alex Sosnovshchenko • 2 years ago
>
> Спасибо!
>
> 2 ^ | ∨ • Reply • Share ›

> **chrisolah** Mod ➜ Alex Sosnovshchenko • 2 years ago
>
> That looks lovely, Alex! Thanks for the translation!
>
> 1 ^ | ∨ • Reply • Share ›

> **Nithin Ts** ➜ Alex Sosnovshchenko • 16 days ago
>
> Did you use LSTM to build a model to translate this article ? If so please share the code
>
> ^ | ∨ • Reply • Share ›

Shimul Hassan Nahid • 2 years ago

Excellent post ...

6 ^ | ∨ • Reply • Share ›

Roman Lutai • a year ago

Very informative and clear material you have made, author, thank you. But where can I find something like this with explaining of backpropagation for lstm? If it is even possible...

3 ^ | ∨ • Reply • Share ›

Zachary Jablons • 3 years 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.

3 ^ | ∨ • Reply • Share ›

> **chrisolah** Mod ➜ Zachary Jablons • 3 years ago
>
> I've added the labels -- thanks for pointing that out!
>
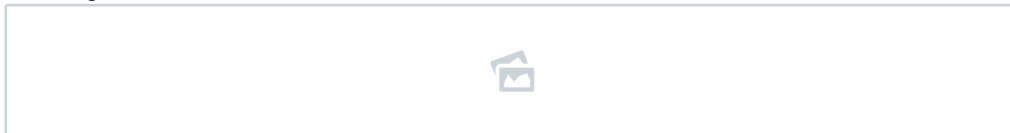> 1 ^ | ∨ • Reply • Share ›

Steven Tang • 3 years ago

Great 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 dimensionsality 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!

5 ^ | ∨ • Reply • Share ›

> **Alfredo Canziani** ➜ Steven Tang • 3 years ago
>
> h_t, C_t and \tilde{C}_t share the same dimensionality.
> The size of \tilde{C}_t comes directly from the 'hight' of W_C matrix (check \tilde{C}_t definition, above).
> W_C's 'width' is instead equal to size(h_t) + size(x_t).
>
> 4 ^ | ∨ • Reply • Share ›

Steven Tang → Alfredo Canziani • 3 years ago

Great, thanks for the explanation!

20 ∧ | ∨ • Reply • Share ›

Majid al-Dosari → Alfredo Canziani • 2 years 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.

1 ∧ | ∨ • Reply • Share ›

> **chrisolah** Mod → Majid al-Dosari • 2 years ago
>
> Here's a figure I made for the post but didn't include in the final version, which may help:
>
>
>
> 11 ∧ | ∨ • Reply • Share ›

> **hughperkins** → chrisolah • 2 years ago
>
> These diagrams look really nice. What are you using to draw them?
>
> 16 ∧ | ∨ • Reply • Share ›

> **Carl Thomé** → hughperkins • 2 years ago
>
> Chris mentioned he did them in Inkscape. Beautiful stuff, indeed!
>
> 2 ∧ | ∨ • Reply • Share ›

> **Majid al-Dosari** → chrisolah • 2 years 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.
>
>
>
> 4 ∧ | ∨ • Reply • Share ›

> **chrisolah** Mod → Majid al-Dosari • 2 years ago
>
> Nice! :)
>
> 2 ∧ | ∨ • Reply • Share ›

> **Artsiom** → chrisolah • a year ago
>
> Hi Chris,
> Could you please tell which software do you use to draw these histograms?
>
> 3 ∧ | ∨ • Reply • Share ›

pluviosilla ➜ chrisolah • 6 months ago

Guess I'm dense, but I still don't see the reason for the extra lines: 2 input lines for x_t, 2 input lines for h_t-1, and two input lines for the state. The simpler diagrams show one input line for each of these three. Why does this diagram sport 6 input lines and precisely 6?

∧ | ∨ • Reply • Share ›

Evalds Urtans ➜ chrisolah • 8 months ago

What did you use to draw this diagram?

∧ | ∨ • Reply • Share ›

Steve_Ohr ➜ Evalds Urtans • 8 months ago

Not what he used, but Omnigraffle will do this for Macintosh users. It's excellent for diagrams.

∧ | ∨ • Reply • Share ›

Evalds Urtans ➜ Steve_Ohr • 8 months ago

Thank you, I will check it out!

1 ∧ | ∨ • Reply • Share ›

Steve_Ohr ➜ Evalds Urtans • 8 months ago

You can download it and use it for free for 30 days.

Objects have 'magnets' that lines snap to. There are pre-defined magnet locations (e.g. 2 each side) but if you hold down the 'm' key you can drag them around and change where lines attach. Lines can hope over/under other lines to and have properties like arrows, dashes, that you can give them.

Great program for any kind of diagramming.

1 ∧ | ∨ • Reply • Share ›

Chris Anderson ➜ chrisolah • a year ago

For that diagram, why is the input and hidden state split in two? I can see why the cell state is. Also, how/why do the yellow layers reduce the 4 inputs to 2 outputs?

∧ | ∨ • Reply • Share ›

Load more comments