Lecture 21
Recurrent Neural Networks

25 April 2016

Taylor B. Arnold
Yale Statistics
STAT 365/665

Yale

Notes

- Problem set 6 was handed back yesterday
- Problem sets 7 & 8 will be returned by Thursday
- Problem set 9 is due a week from today

Recurrent neural networks

Recurrent neural networks address a concern with traditional neural networks that becomes apparent when dealing with, amongst other applications, text analysis: the issue of variable width inputs.

Recurrent neural networks

Recurrent neural networks address a concern with traditional neural networks that becomes apparent when dealing with, amongst other applications, text analysis: the issue of variable width inputs.

This is also, of course, a concern with images but the solution there is quite different because we can stretch and scale images to fit whatever size we need at the moment. This not so with words.

RNNs as time

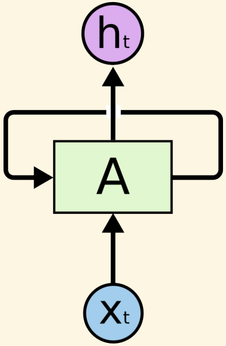
An equivalent framing of this problem is to think of a string of text as streaming in over time. Regardless of how many words I have seen in a given document, I want to make as good an estimate as possible about whatever outcome is of interest at that moment.

Stateful models

Using this idea, we can think of variable width inputs such that each new word simply updates our current prediction. In this way an RNN has two types of data inside of it:

- fixed weights, just as we have been using with CNNs
- stateful variables that are updated as it observes words in a document

We can also think of this as giving ‘memory’ to the neural network.



A third way of thinking about recurrent neural networks is to think of a network that has a loop in it. However, the self-input get's applied the *next* time it is called.

A fourth way of thinking about a recurrent neural network is mathematically. We now have two parts to the update function in the RNN:

$$h_t = Wx_t + b + Uh_{t-1}$$

A fourth way of thinking about a recurrent neural network is mathematically. We now have two parts to the update function in the RNN:

$$h_t = Wx_t + b + Uh_{t-1}$$

Notice that U must always be a square matrix, because we could unravel this one time further to yield:

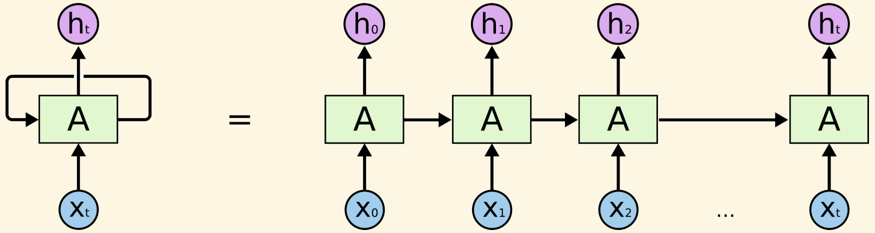
$$h_t = Wx_t + b + UWx_{t-1} + Ub + U^2 h_{t-2}$$

A note on time initialization

One confusing bit, at least for me the first time I saw RNNs, is the relationship between time and samples. We typically restart the state, or memory, of the RNN when we move on to a new sample. This detail seems to be glossed over in most tutorials on RNNs, but I think it clarifies a key idea in what these models are capturing.

Unrolling an RNN

In truth, an RNN can be seen as a traditional feedforward neural network by unrolling the time component (assuming that there is a fixed number of time steps).



Unrolling the recurrent neural network.

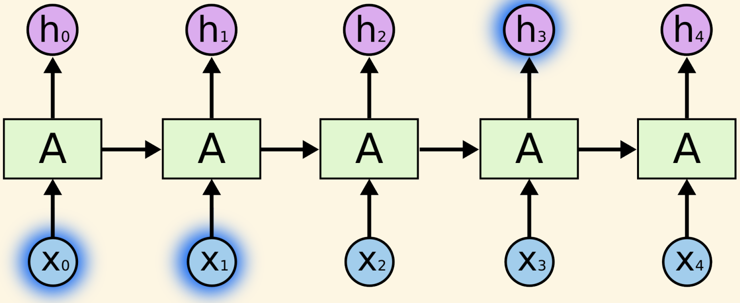
Training RNNs

While it is nice that we get a ‘running output’ from the model, when we train RNNs we typically ignore all but the final output to the model. Getting the right answer after we have looked at the entire document is the end goal, anyway. To do this, back-propogation can be used as before.

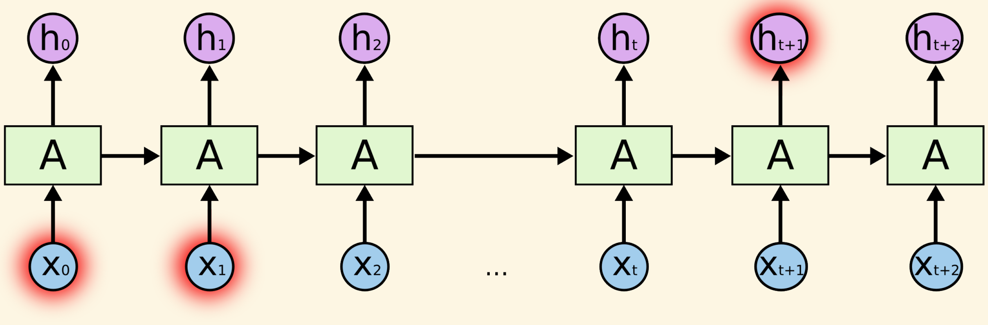
While we could unroll the RNN into a FF network and apply the algorithms we saw in Lecture 13, for both memory consumption and computational efficiency, techniques exist to short-cut this approach.

I. Load IMDB dataset

II. Basic RNN example



Because of the state in the model, words that occur early in the sequence can still have an influence on later outputs.



Using a basic dense layer as the RNN unit, however, makes it so that long range effects are hard to pass on.

Long short-term memory was original proposed way back in 1997 in order to alleviate this problem.

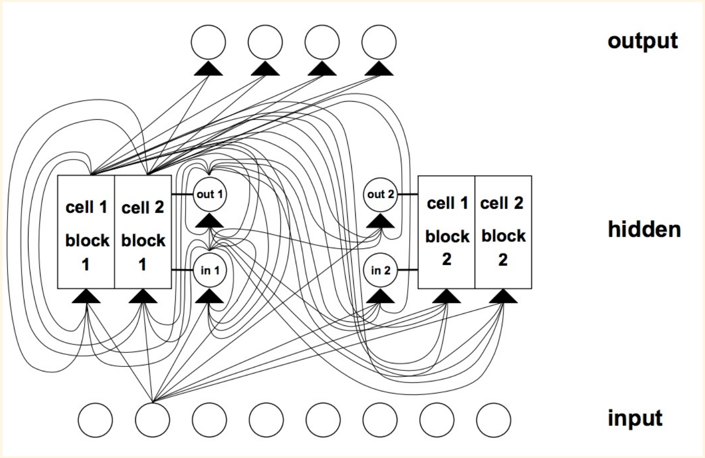
Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9, no. 8 (1997): 1735-1780.

Their specific idea that has had surprising staying power.

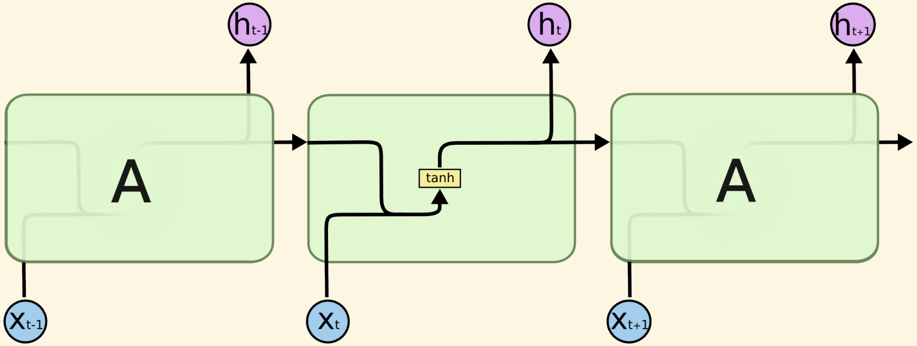
A great reference for dissecting the details of their paper is the blog post by Christopher Olah:

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

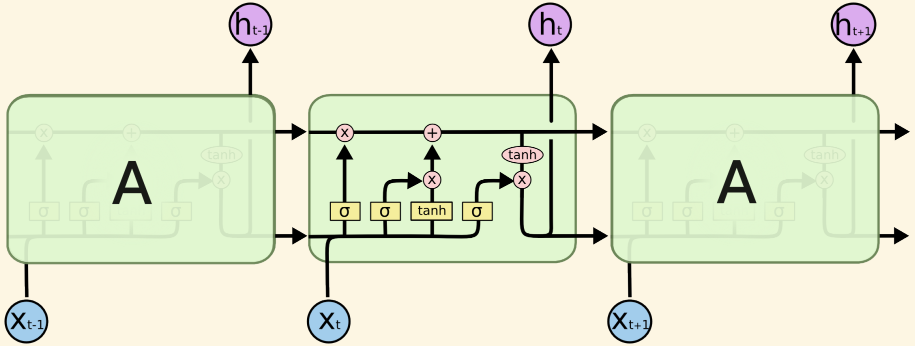
I will pull extensively from it throughout the remainder of today's lecture.



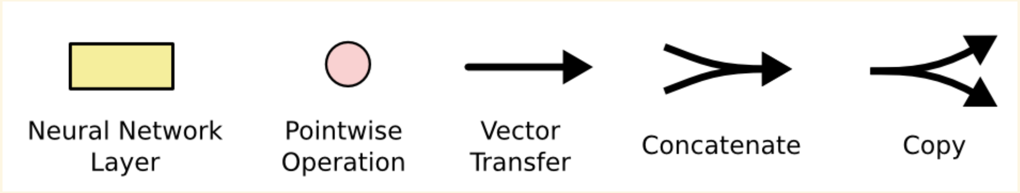
Some people consider LSTM's to be a bit hard to understand; here is a diagram from the original paper that partially explains where the confusion comes from!



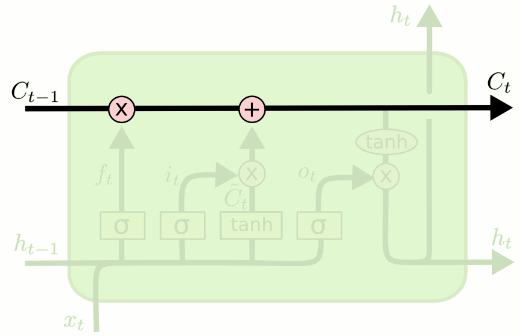
In fact, though, basic idea of an LSTM layer is exactly the same as a simple RNN layer.



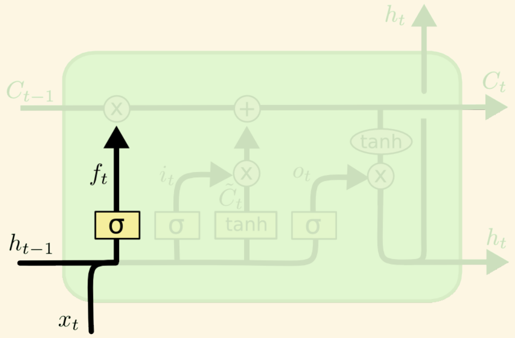
It is just that the internal mechanism is just a bit more complex, with two separate self-loops and several independent weight functions to serve slightly different purposes.



The diagrams use a few simple mechanics, most of which we have seen in some form in CNNs. The pointwise operation, for example, is used in the ResNet architecture when creating skip-connections.

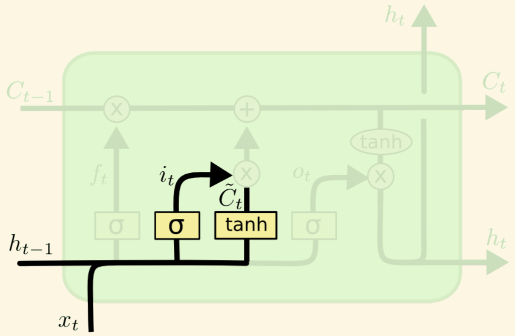


A key idea is to separate the response that is passed back into the LSTM and the output that is emitted; there is no particular reason these need to be the same. The **cell state** is the part of the layer that get's passed back, and is changed from iteration to iteration only by two linear functions.



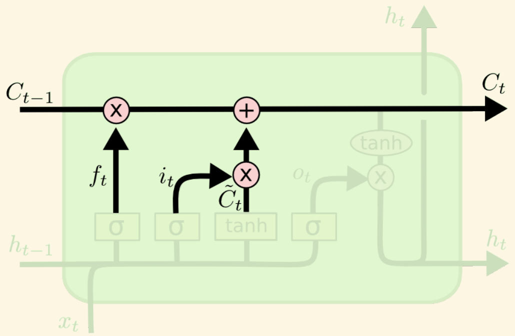
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Next, consider the **forget gate**. It uses the previous output h_{t-1} and the current input x_t to determine multiplicative weights to apply to the cell state. We use a sigmoid layer here because it makes sense to have weights between 0 and 1.



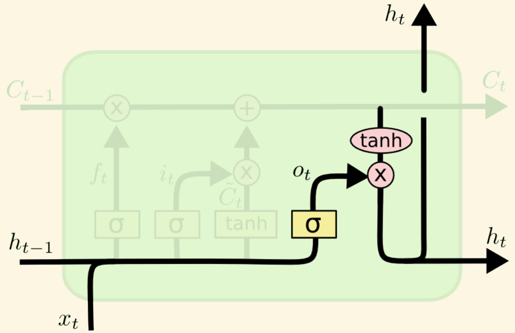
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Next, we have a choice of how to update the cell state. This is done by multiplying an input gate (again, with a sigmoid layer) by a tanh activated linear layer.



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

The cell state of the next iteration is now completely determined, and can be calculated directly.

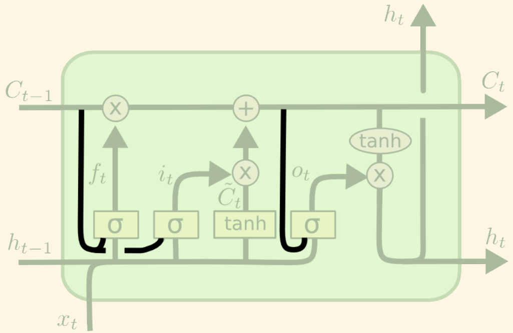


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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

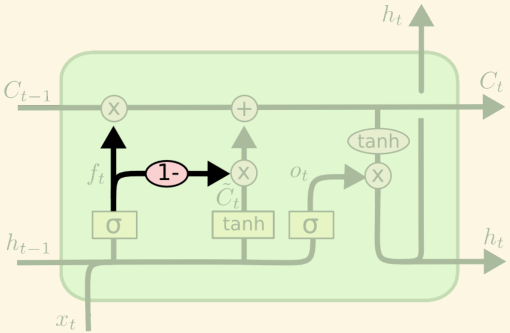
Now, to determine the output of the model, we want to emit a weighted version of the cell state. This is done by applying a tanh activation and multiplying by the fourth and final set of weights: the output weights. This passed both as an output to the LSTM layer as well as into the next time step of the LSTM.

III. LSTM



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

Over the years, variants on the LSTM layers have been given. Confusingly, these are often presented *as* LSTM layers rather than minor variants on the original technique. One modification is to add **peepholes** so that the input, forget, and output gates also take the current cell state into account.



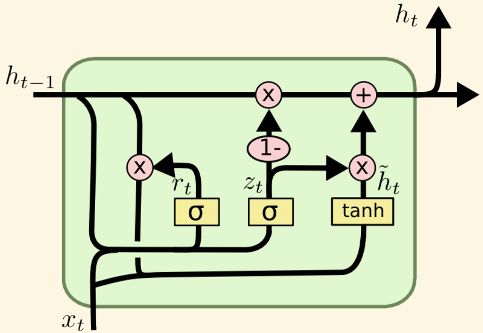
$$C_t = f_t * C_{t-1} + \boldsymbol{(1 - f_t)} * \tilde{C}_t$$

One natural extension is to set the input and forget gates to be the negation of one another.

A more dramatically different alternative is known as a Gated Recurrent Unit (GRU), originally presented in this paper:

Cho, Kyunghyun, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. "On the properties of neural machine translation: Encoder-decoder approaches." arXiv preprint arXiv:1409.1259 (2014).

One benefit is that is offers a slight simplification in the model with no systematic performance penalty. Along with LSTM, it is the only other model implemented in keras, which should point to its growing popularity.



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

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

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

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

In short, in combines the input and cell states together, and combines the forget and input gates. This results in one fewer set of weight matrices to learn.

If you would like a good, comphrensive, and empirical evaluation of the various tweaks to these recurrent structures, I recommend this paper

> *Greff, Klaus, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. "LSTM: A search space odyssey." arXiv preprint arXiv:1503.04069 (2015).*

As well as this article:

> *Jozefowicz, Rafal, Wojciech Zaremba, and Ilya Sutskever. "An empirical exploration of recurrent network architectures." In Proceedings of the 32nd International Conference on Machine Learning (ICML-15), pp. 2342-2350. 2015.*

Though, once you fully understand the LSTM model, the specifics amongst the competing approaches typically do not require understanding any new big ideas.

IV. GRU

V. Evaluating a sequence of inputs

VI. Visualize the output