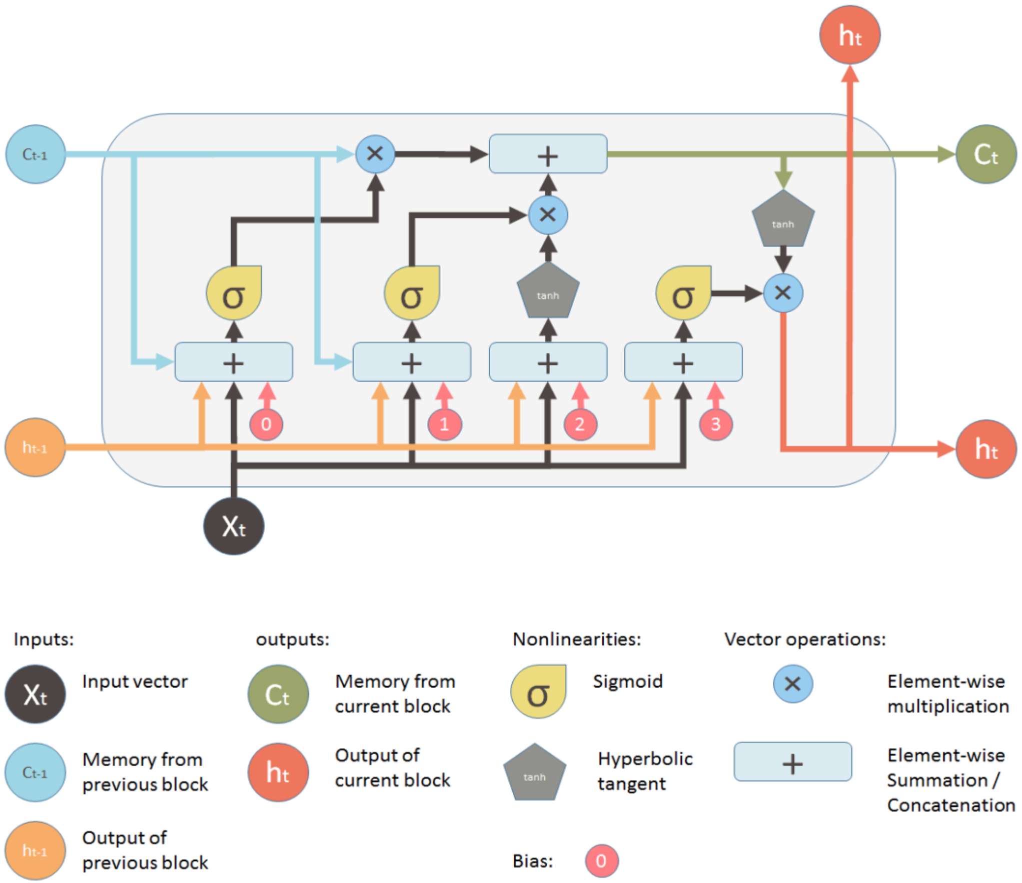
Nested LSTM Intuition

Nested LSTMs (NLSTMs) are a form of LSTMs, but the main difference is that instead of stacking cells, NLSTM cells are nested.

If we look at a traditional LSTM,

```
[ ]  %%html
     <img src='/nbextensions/google.colab/lstm.png' / style="width:700px;height:600px;" / alt="Traditional LSTM">
```



LSTM Step-by-Step Approach

Forget Gate

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."

In the sense of click through rates, if it sees a different item-id, say 5, from the data previously, it the forget gate should "forget" all of the data with the previous item_id's, except for id=5. Mathematically, the forget gate is defined as

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

Input Gate

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, 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.

For click rate, we want to add the item_id 5 to the cell state to replace the old item_ids. The input gate is defined as

$$i_t = \sigma(W_i) \cdot [h_{t-1}, x_t + b_i]$$
$$C_t^* = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Current Cell Gate

Now, we apply the forget gate and input gate's calculations of previous time steps of $C_{t-1,t-2,\dots,t-T}$ onto the current cell C_t.

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

Current cell gate is defined as

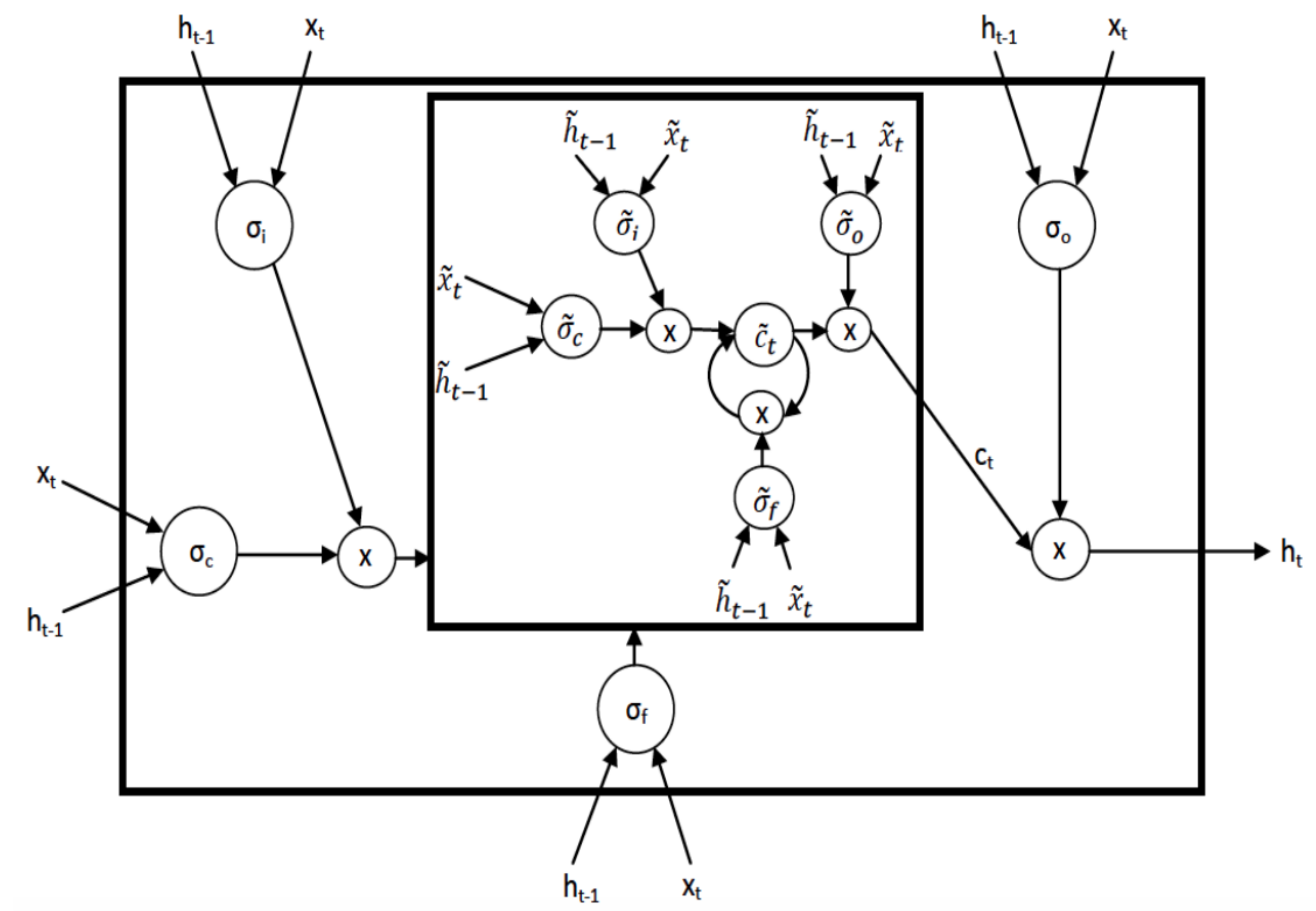
$$C_t = f_t C_{t-1} + i_t C_t^*$$

Output Gate

This output will be based on our cell state, but will be a filtered version. First, we run a $\mathrm{sigmoid}$ 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 $\mathrm{sigmoid}$ gate, so that we only output the parts we decided to.

In the click-through ad prediction, we are transforming C_t and creating x_t for the next cell's forget gate. The output gate is defined as

$$o_t = \sigma(W_0 \cdot h[t_1, x_t] + b_o)$$
$$h_t = o_t \cdot \tanh(C_t)$$



Nested LSTMS

Nested LSTMs nest cells instead of stacking them, so the main difference is the ways in which h_{t-1}, x_t, and C_t are computed.

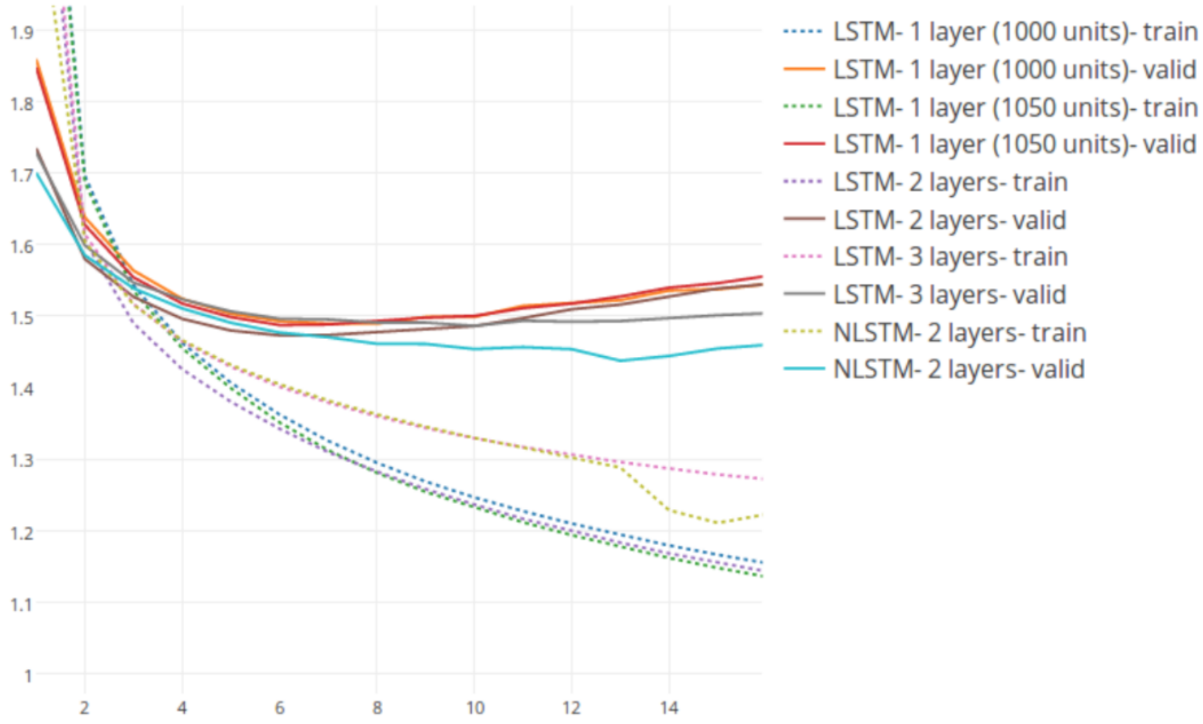
Instead of finding C_t through an additive method, NLSTMs use a learned function, denoted as m that is the "inner memory" of the NLSTM. This means that the NLSTM pulls from the cell C_{t-1} that is nested within C_t.

$$
\begin{aligned}
C_t &= f_t C_{t-1} + i_t C_t^* && \textbf{LSTM} \\
C_t &= m(f_t Ct_1, i_t C_t^*) && \textbf{NLSTM}
\end{aligned}
$$

Performance

A test on predicting poems of different time periods shows various implementations of layered LSTMs and NLSTMs.

```
[ ]  %%html
     <img src='/nbextensions/google.colab/performance.png' / style="width:800px;height:500px;" / alt="Traditional LSTM">
```



This graph shows the log-loss error function over time. Solid lines are the validation set, and dotted lines are the test set.

We can see that a two-layer NLSTM, meaning a LSTM within a LSTM has the lowest log-loss error function out of LSTMs with layers up to 3.

Clearly, the NLSTM is more accurate than LSTMs, so I used it to predict sales prices, given the sales_id, store_id, item_cnt_day, item_price. I was trying to predict the item_cnt_day for all items one month in the future by using an NLSTM on the time-series data from 2013-2015. The item_cnt_day counts the number of items with specified sales_id sold for that day.

Conceptually, a higher item_cnt_day would result in more sales for the item with the specified id. Thus, the seller can prioritize this item.

However, the data is for physical stores, not online advertisements, which tend to focus more on increasing click-through rate. Currently, click-through rate for an online advertisement is around 17%, but specific ads can be more or less effective. Also, I was not able to calculate the sales percentage rate per day yet, so currently my output shows one month's worth of ending inventory for a given item. More work has to be done to analyze the output.

Sources:

- https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://arxiv.org/pdf/1801.10308.pdf
- https://github.com/hannw/nlstm
- https://github.com/titu1994

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day
0	2013-01-02	0	59	22154	999.00	1.0
1	2013-01-03	0	25	2552	899.00	1.0
2	2013-01-05	0	25	2552	899.00	-1.0
3	2013-01-06	0	25	2554	1709.05	1.0
4	2013-01-15	0	25	2555	1099.00	1.0

	ID	shop_id	item_id
0	0	5	5037
1	1	5	5320
2	2	5	5233
3	3	5	5232
4	4	5	5268

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
nested_lstm_2 (NestedLSTM)   (None, 40)                19680
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 41
=================================================================
Total params: 19,721
Trainable params: 19,721
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/5
214200/214200 [==============================] - 97s 452us/step - loss: 30.1095 - mean_squared_error: 30.1095
Epoch 2/5
214200/214200 [==============================] - 96s 447us/step - loss: 29.9201 - mean_squared_error: 29.9201
Epoch 3/5
214200/214200 [==============================] - 95s 441us/step - loss: 29.7502 - mean_squared_error: 29.7502
Epoch 4/5
214200/214200 [==============================] - 93s 433us/step - loss: 29.6220 - mean_squared_error: 29.6220
Epoch 5/5
214200/214200 [==============================] - 93s 432us/step - loss: 29.5169 - mean_squared_error: 29.5169
```

	item_cnt_month
0	0.517290
1	0.087915
2	0.834995
3	0.101971
4	0.087915
5	0.453989
6	0.954998
7	0.103927
8	1.264257
9	0.087915
10	3.159820
11	0.142026
12	0.088059
13	0.390866
14	1.697171
15	3.045769
16	0.087915
17	0.094506
18	1.443809
19	0.088286
20	0.640181
21	0.087915
22	0.659080
23	0.725886
24	1.530892
25	0.087915
26	0.087915
27	0.509684
28	0.838930
29	4.274371