Language Models (Feed Forward and RNN)

Jonathan May

September 6, 2023

1 Feed-forward language models

We previously introduced Multilayer perceptrons (MLPs) for classification. How can we use these for language modeling? Further, why might these be useful for language modeling?

First the how. Although this can be multi-layered, let's use a simple one-layer architecture, assume a context of four words, and a non-linear function for the hidden layer called g. We assume the dimension of a word representation (aka the 'embedding') is 50 and the dimension of the hidden representation is 100. We also assume a vocabulary of 20,000 words. When calculating the hidden and output vectors (but not the embedding) we will assume a bias term. We thus have the following weight (i.e. parameter) matrices, which are listed along with their dimensions:

- Embedding table E : (20,000 x 50)
- Hidden weights H : (200 x 100)
- Hidden bias b_H : (1 x 100)
- Output weights O : (100 x 20,000)
- Output bias b_O : (1 x 20,000)

E is indexed by the vocabulary. We want to get the probability of the example above, in a 5-gram model. So that is P(Call|START,START,START,START) P(me|Call,START,START,START)P(Ishmael|me,Call,START,START), etc. For the first term, P(Call|START,START,START,START), assume we have some row in E for START. Concatenate four copies of that row; we'll call that x. Put that into the network and then when we get to softmax and the output probabilities, we retrieve the cell for `Call`. Then we continue similarly for P(me|Call,START,START,START) and so on.

What is the advantage to using this approach instead of the non-neural n-gram approach (apart from the empirical superior performance)?

Smoothing/Interpolation: Notice that for any 4-gram over the vocabulary space we can get the probability of each word. There is no explicit backoff or smoothing here.

Storage: Let's compare the sizes of the feedforward and non-feedforward models. First sum up the feedforward:

- E: 1,000,000
- $H + b_H$: 20,100
- $O + b_O$: 2,020,000
- Total: 3,040,000

Notice that this is not dependent on how much training data we use. It is dependent on some modeling decisions, like embedding and hidden size, number of layers, etc. Most of all it's dependent on vocabulary. Contrast this with the number of parameters used for a 5-gram model. This is highly dependent on corpus size. Below we show the parameter size if the model is trained on moby dick and tom sawyer and compare this to training on the treebank portion of the wall street journal. Note that both are considered fairly small corpora for language modeling. I also added stats for gigaword, a much larger corpus.

n-gram	moby dick + tom sawyer	wsj treebank	gigaword
(tokens)	181,017	1,107,391	4.2b
5	180,612	1,074,244	558m
4	178,904	1,037,648	447m
3	167,372	906,848	246m
2	113,439	530,884	60m
1	27,279	85,967	1m
total	667,606	3,635,591	1.3B

The feedforward model, meanwhile, remains a constant size.

1.1 Why should this work?

To some degree this is a bit of a mystery on a deep level. But there is some good intuition for why this might work. There are multiple angles to address this question; here's one:

A word embedding can be thought of as features specific to that word type. Rather than treat each of 20k words as 20k independent items, we characterize them as having certain properties. We could do this by hand (e.g. 'French origin', 'animate', 'plural') but rely on training for these features to be implicitly defined. By representing a 20,000-dimensional object in 50-space, some properties shared among disparate objects have to be identified, otherwise learning won't happen.

Similarly, the hidden representation is a 100-dimensional representation that generalizes a four-gram – there are $20,000^4$ possible such objects. This is formed by considering multiple embedding features together simultaneously. If we added more hidden layers this would consider multiples of multiples of features simultaneously.

2 Limitations of Feed-Forward Networks

Feed-forward language models solve a lot of the problems with (non-neural) n-gram models. Specifically they

- generalize, handling the sparsity problem much better than n-gram models, with much lower perplexity on unseen n-grams.
- are efficient, requiring constant memory and do not blow up with the amount of training data (and hence noverl n-grams) seen.
- allow for larger n; beyond 5-grams, non-neural models were never that helpful. We used 12-gram feed forward models effectively.

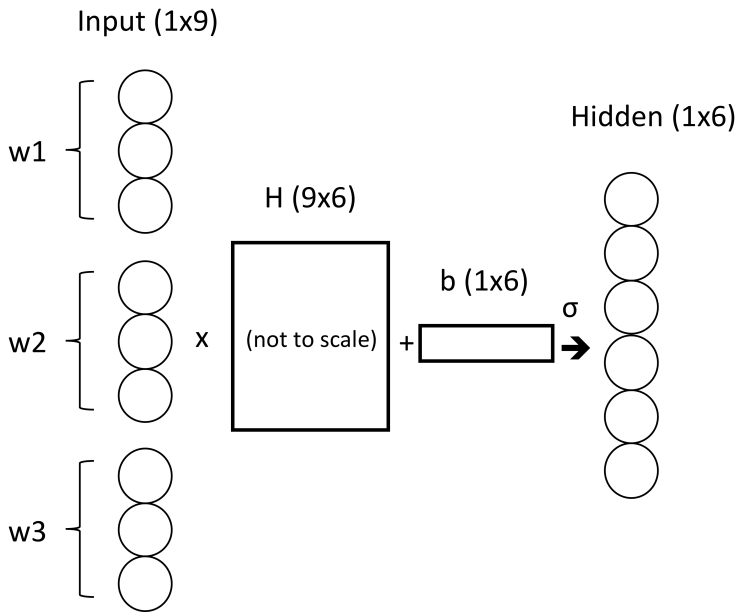
There are some limitations, however. Some are newly introduced, while some are persistent and may now be addressed.

- Although the size of feed-forward is fixed, it is highly dependent on vocabulary size (this also determines computation time). This limits the vocabulary rather significantly. There have been methods to overcome this, either partially or totally, some of which we will talk about (subword models) and some of which we won't (noise-contrastive estimation, hierarchical softmax).
- Compared to non-neural models, neural models take a rather long time to train, since parameter estimation using maximum likelihood and smoothing requires only one pass through the data, (with very simple calculations). The use of GPUs helps this somewhat but it remains an ongoing concern we won't directly discuss.
- Correctly modeling language can require very long context. E.g. `My mother, who once caught a balance beam in her eye, but is actually nicer than you might expect, lived with three bermuda sharks and (like/likes) ice cream.`. We are fundamentally limited by whatever n we select. We will directly address this now.

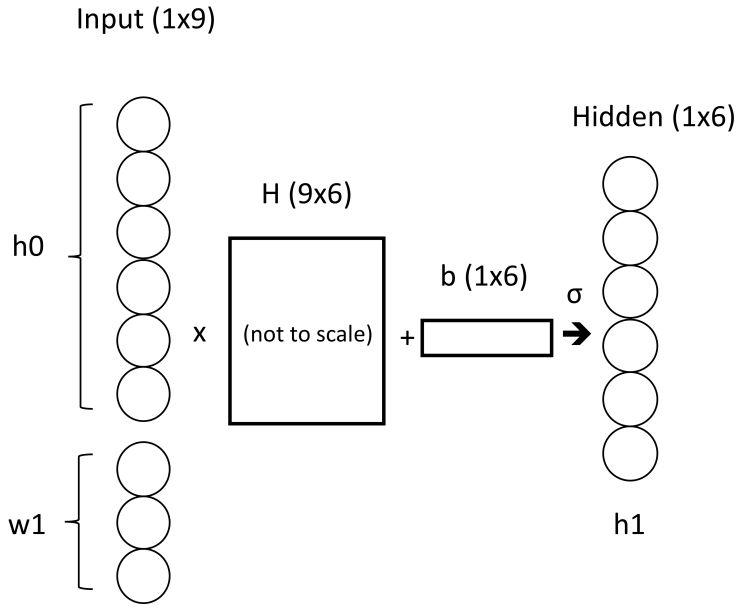
3 Recurrent Neural Networks (RNNs)

In some respects, RNNs are very similar to feed-forward networks. A word representation feeds into one or more hidden layers, fully connected and with a nonlinear function, and ultimately the hidden representation is used to predict the probability of the next word. Gradient descent along the cross-entropy loss via backpropagation is used to update parameters.

The key difference is in the structure of the parameters, specifically in the construction of the hidden vector, h. Here is the way h is constructed in feed-forward, as a 4-gram model with an embedding dimension of 3 and a hidden dimension of 6:

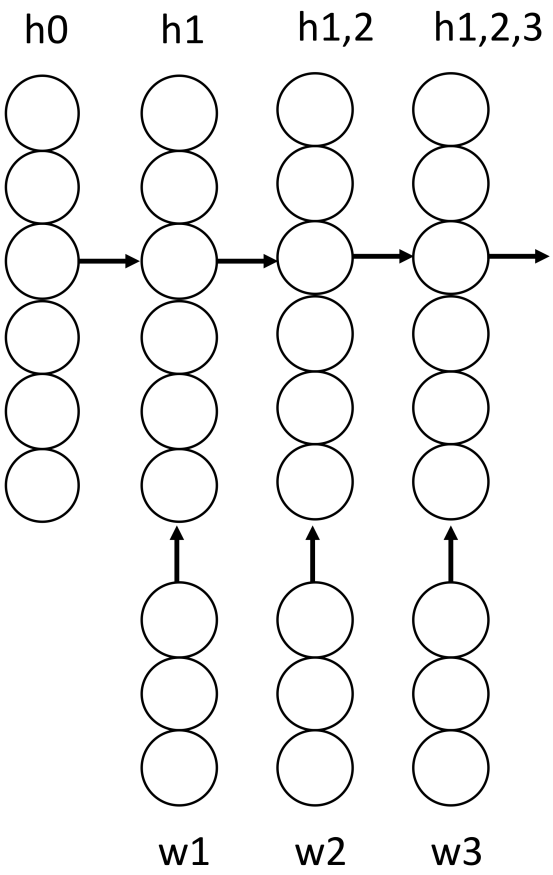


By comparison here is the construction of a hidden unit h_1 for an RNN:



Notice that the input to the RNN is a hidden vector and a word embedding. The output is another hidden vector.[1] However, this only captures the hidden representation for one word. If we want to capture the sequence w_1, w_2, w_3 we simply re-do the calculation, using the last calculated h. Note that at each step we use the *same* H and b_H and these are not shown in the diagram below for space reasons.

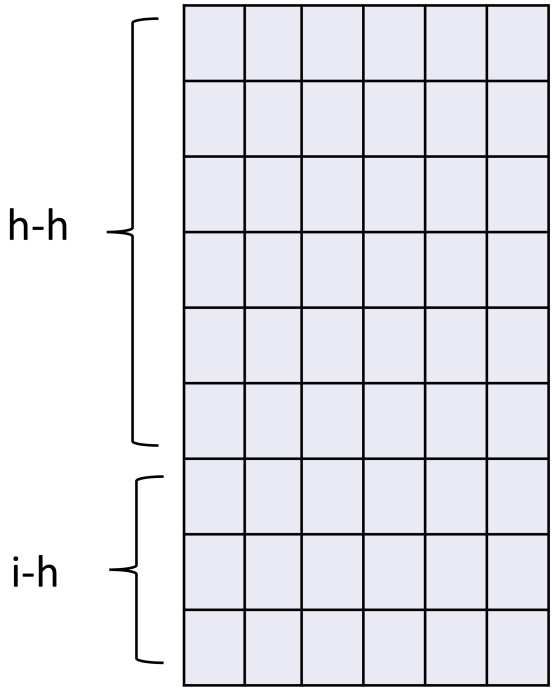
[1]There's no special requirement about the size of the hidden or embedding vectors; I just made them look very similar to illustrate how similar the computation is.



$h_{1,2,3}$ is a representation of w_1, w_2, w_3 the same as the h in the feed-forward example. Just as in the feed-forward case, we can get a logit layer from $h_{1,2,3}$ (which we will now call simply h_3) and then with softmax get probabilities over the next word. But of course we can continue adding words and getting new hidden vectors that capture more and more context.

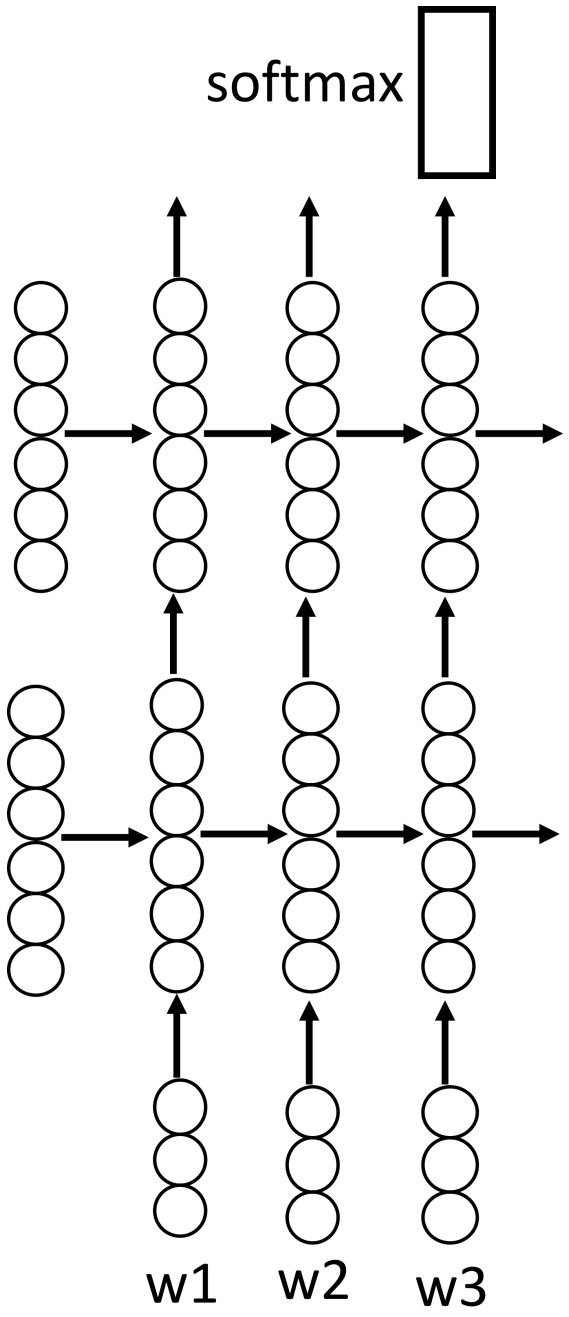
The hidden vectors can be viewed as (and I often describe them as) *states* in the sense of a finite-state automaton. Each position in hidden space is a state, from which an arc labeled with each word in the vocabulary leads to another state. Different from an FSA, however, is that an RNN is an *infinite*-state automaton (at least to the level of representability in hardware).

Note that H can be divided into the piece that applies to the hidden state input and the piece that applies to the lexical input; the pieces are frequently written as H_{ih} and H_{hh} to distinguish between the weights relevant to the context (hidden-to-hidden) and those relevant to the words (input-to-hidden):



Recall that feed-forward parameters were dependent on the amount of context that was used; for each additional word of context there were Vxhidden more parameters; as V can be large this is quite substantial.

As with feed-forward networks, RNNs can be *stacked*; this is where the 'deep' in deep learning comes from. The hidden unit at a layer becomes the 'input' to the next layer. Typically, the last layer is then converted (via output weights) to logits to predict the next word in the sequence. Typically, each layer has a separate learned h_0, H, and b_H, which are learned, as is the output weight matrix O and bias b_O. So if the non-linear function is σ, the embedding of word w_j is x_j, the hidden weights at layer i are H_i, $b_{H,i}$ and the initial hidden state $h_{0,i}$, the logits for predicting w_2 would be calculated as $\sigma([\sigma([x_1; h_{0,1}]H_1 + b_{H,1}); h_{0,2}]H_2 + b_{H,2})O + b_O$.



3.1 Training

The structure of the training for RNN LMs is quite simple; you constantly evaluate the prediction of the next word given the last context. In theory this is done over the entire corpus, but in practice this can be very slow, since gradients are calculated with respect to each step in the context, so context is limited to the sentence. Standard cross-entropy loss is generally used.

4 Unreasonable effectiveness of

it turns out RNNs are pretty powerful generation models; this was not always true of the strictly n-gram models. Andrej Karpathy, in a blog post titled 'The Unreasonable Effectiveness of RNNs'[2] built an RNN that operated one *character* at a time and trained it on various kinds of text, then generated samples from the trained models and got surprisingly good outputs.

4.1 Shakespeare

```
PANDARUS:
Alas, I think he shall be come approached and the day When little srain would be attain'd into being never fed, And who is but a chain and subjects of his death, I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul, Breaking and strongly should be buried, when I perish The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord: They would be ruled after this chamber, and my fair nues begun out of the fact, to be conveyed, Whose noble souls I'll have the heart of the wars.

Clown: Come, sir, I will make did behold your worship.

VIOLA: I'll drink it.
```

4.2 Wikipedia

```
== ''Declaration of Protectance from Iceland'' ==

In the late 1970s, [[Deep Seols]] and the Australian Federal Navy in order to establish a police duty of a several federal government of the world.

Since 2004 the state regarded as a [[Suffolk Act 1994]], but the [[Army personality|Armed Forces]] appeared in Paris, despots with Nelson concentrated on) was inaugurated as his father. Heraldry put an attempt to get influent territory register. Hayling among their lost operations, a population of Deliberate countries arrived and Harry Elser, established [[The West Virasian Socialist Wars]] for 16 year and modern democratic 30 [[Justice|booms]] elections to the CDC.
```

[2] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

4.3 LaTeX (some compile bugs had to be fixed)

For $\bigoplus_{n=1,\dots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

5 Other Uses

- Tagging (POS, NE); generate a label at every word
- Sentence classification: e.g. sentiment. Take some hidden state as the 'sentence' state – could be the last, could be an average, then predict class label.
- As a sentence representation in something more complicated (e.g. question answering)
- Generation given context (e.g. speech recognition, machine translation)

In many of the whole-sentence variants above a *bidirectional RNN* is used; I'll give a quick sketch but we'll see this more in MT and IE lectures.

6 Variants

As we have seen, RNNs can in theory represent infinite context. In practice it is hard to do this due to some practical considerations. Calculating gradient through time requires repeated multiplications of the hidden weight matrix. It turns out that if the largest eigenvalue of $H < 1$ then the gradient will shrink exponentially (vanish), which means after not many words of context you won't see any effect. Also, if the largest eigenvalue of $H > 1$ then the gradient will grow exponentially (explode) which can cause updates to be too large or even NaN.

Exploding gradients can be clipped: define a maximum gradient amount (i think '5' is often used) and if the L_2 norm exceeds that amount, divide the gradients by $max/||g||$. For

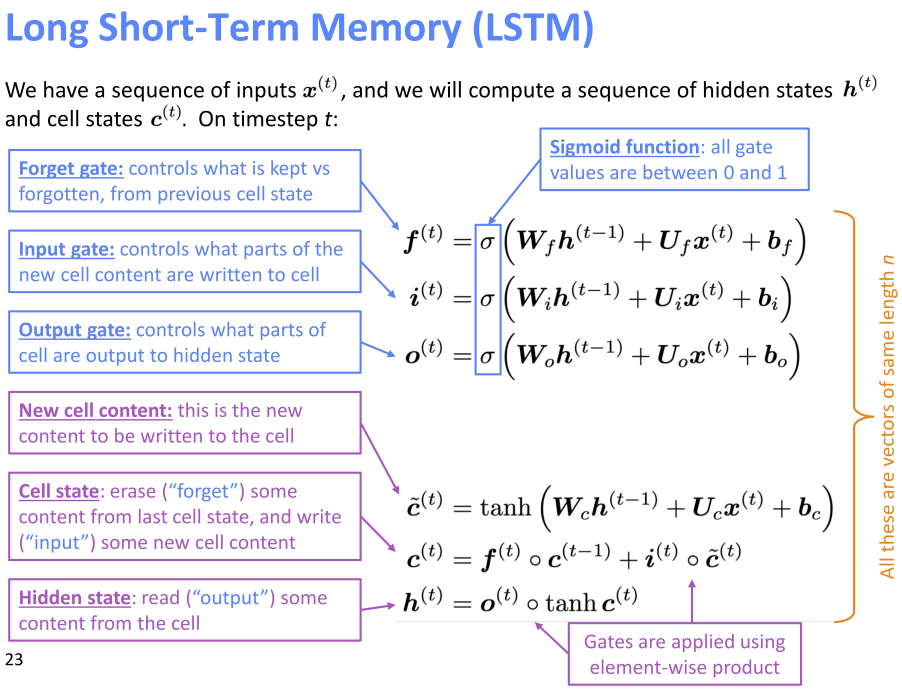
vanishing gradients scaling can be used, but another solution was found: a more complicated RNN that has a *memory* element and a means of learning how much memory to keep from step to step.

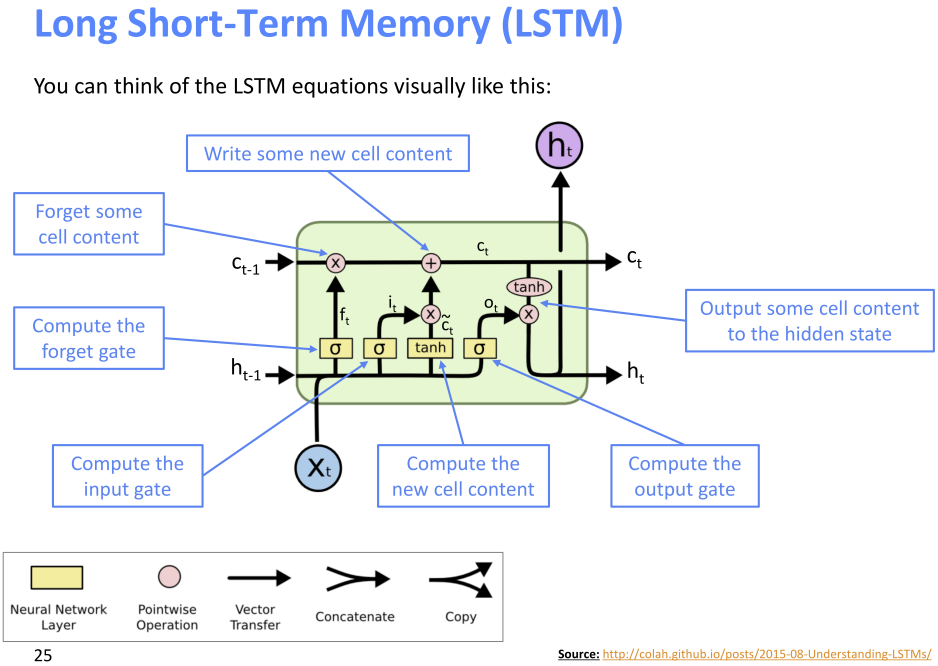
6.1 Long Short-Term Memory (LSTM)

The key to LSTM is as follows:

- The 'pre-cell' is a standard nonlinear calculation (typically tanh or Relu).
- Input, forget, and output 'gates' are vectors of values from 0 to 1 (typically sigmoid)
- The cell is formed by gating how much should be input from the pre-cell and how much should be forgotten from the last cell (then adding these).
- The hidden state is formed by tanh-ing the cell and then using the output gate to determine how much gets through.

The equation slide from Abi See (stanford class) and figure from Chris Olah (also used by Abi See) help me understand.





Let's walk through it. The gates themselves are all identical. Imagine if this was a generic RNN. Then the hidden state would be $g(U(x_t; h_{t-1}))$ where g is the nonlinear function, x_t is the data input and h_{t-1} is the last state. Each gate uses a sigmoid function to go from 0 to 1 and has its own U; one for forget (f), one for input (i), one for output (o).

Let's also assume not only is there an incoming hidden state h_{t-1}, there's a different kind of incoming hidden state called the *cell*, c_{t-1}. First we do $f \times c_{t-1}$, which means some of the cell state gets forgotten.

Then we want to add in some new content to the cell. We have a U for that, too, specified for the cell state. In LSTM, the nonlinear function is tanh. So we form the preliminary new cell state $\tilde{c}_t = \tanh(U_c(x_t; h_{t-1}))$. But we only want to keep some of that in the cell state so we multiply it by the input gate. Then the new cell state is $f \times c_{t-1} + i \times \tilde{c}_t$, keeping some of the old cell and some of the new data.

What about h_t? We first squash c_t through tanh and then use the output gate to determine how much of that to keep. $h_t = o \times \tanh(c_t)$.

6.2 Gated Recurrent Units

Very similar idea to LSTMs; proposed by Cho et al. in 2014. Simnpler than LSTM but same idea:

- No cell state. Two gates; update and reset.
- pre-hidden state calculated like a classic hidden state but hadamard of reset with the previous hidden state (like a forget gate)
- final hidden state interpolates between last hidden and current pre-hidden using update and 1-update (which thus functions like an input and output/forget

Again, the slides:

Gated Recurrent Units (GRU)

- Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
- On each timestep t we have input $\boldsymbol{x}^{(t)}$ and hidden state $\boldsymbol{h}^{(t)}$ (no cell state).

Update gate: controls what parts of hidden state are updated vs preserved

$$\boldsymbol{u}^{(t)} = \sigma\left(\boldsymbol{W}_u\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u\boldsymbol{x}^{(t)} + \boldsymbol{b}_u\right)$$

Reset gate: controls what parts of previous hidden state are used to compute new content

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r\boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right)$$

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

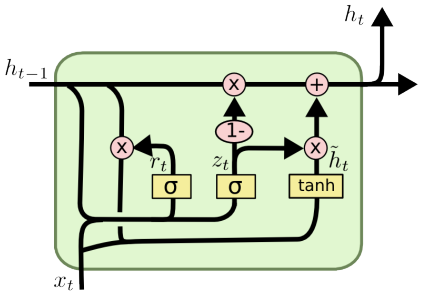
$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h(\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}_h\boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$

How does this solve vanishing gradient?
Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)

28 "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", Cho et al. 2014, https://arxiv.org/pdf/1406.1078v3.pdf



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

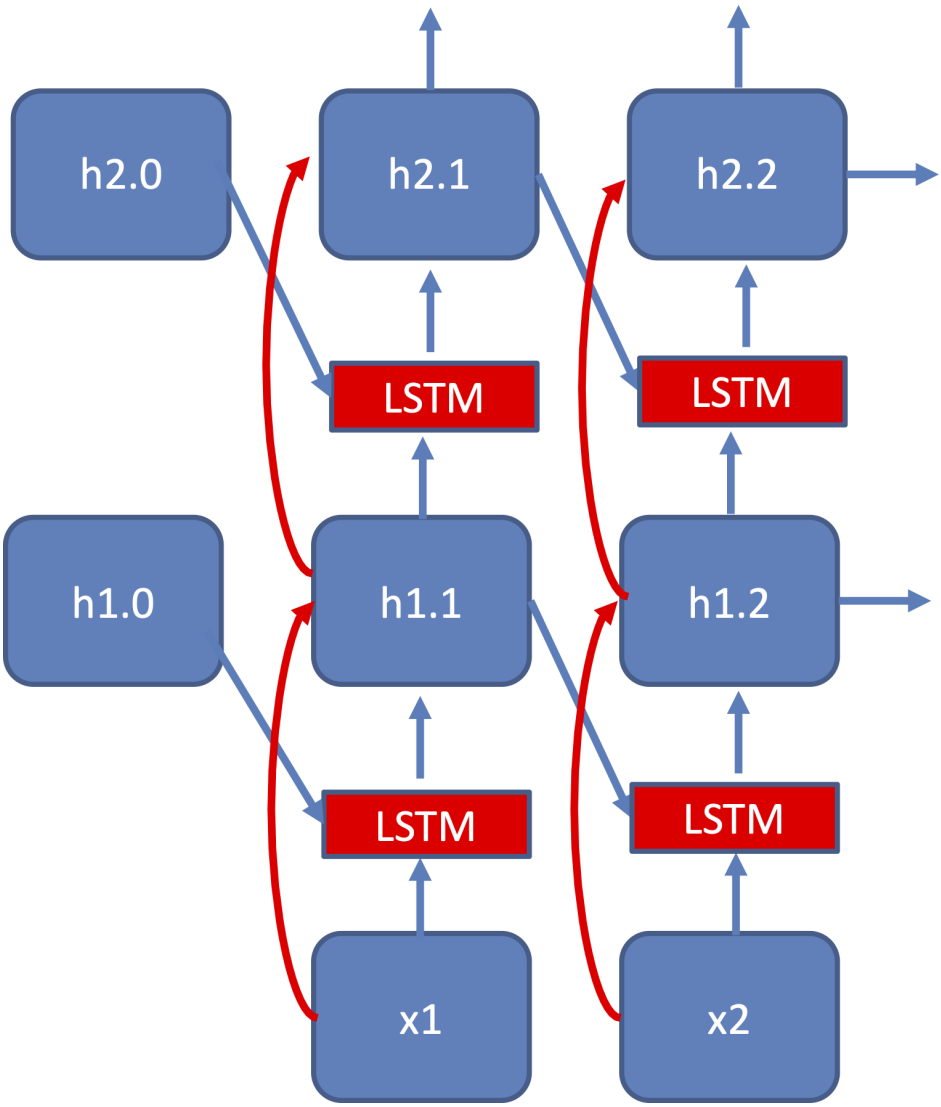
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

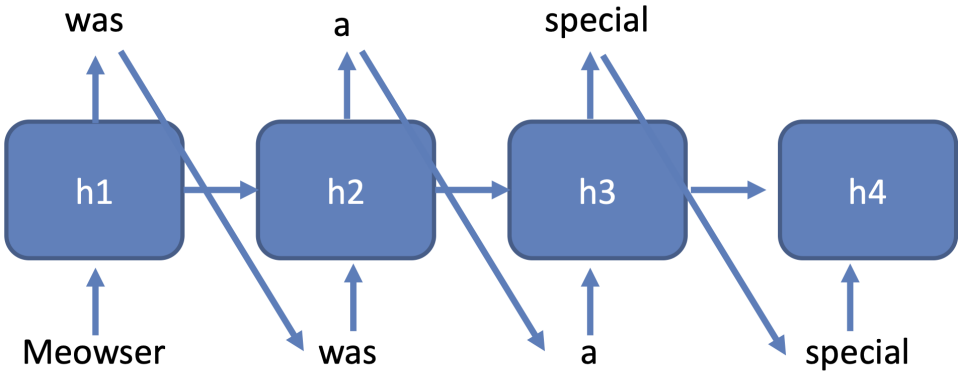
There are other variants but LSTM and GRU are the most widely used RNNs. It's unclear which is better. Other ways to prevent forgetting the past are the use of *attention* and *residual connections*, which are both ways of 'skipping' the chain of words in one way or another.

6.3 Residuals



Simplifying the equation above, let's let $h_1 = \sigma([x_1; h_{0,1}]H_1 + b_{H,1})$ and $h_2 = \sigma(h_1; h_{0,2}]H_2 + b_{H,2})$. However, when going up a layer, add the last layer in on top of the computed layer. So $h_2 = h_1 + h_2$. Visually it skips the processing of the layer to ensure nothing gets 'diluted.'

6.4 Attention



Consider the figure above. We want to predict the word after 'special.' The most informative word for predicting that is not 'special', it's 'Meowser.' State h4 contains information about all the previously seen words but the ones seen a while ago are less 'fresh' (empirically). Instead of predicting the next word from just h4, let's predict it from a weighted combination of all previous states. But how much should we **attend** to each state? Well, of course we'll naturally learn that!

Let the output embeddings matrix be $O(u \times |V|)$. Without attention, the word choice is $\text{argmax}\, Oh_4$. Let's introduce a matrix $A(u \times u)$ for converting each of our previous hidden states $h_1 \dots h_4$ into vectors with the special task of determining relatedness.[3] $Ah_1 = a_1$ and so on for $a_2 \dots a_4$. Then $a_1 \cdot h_4$ is a scalar, and its magnitude will be higher if a_1 and h_4 are more similar. We can use softmax across $[a_1 \cdot h_4, a_2 \cdot h_4, a_3 \cdot h_4, a_4 \cdot h_4]$ to get a distribution, which we can call $\alpha_1, \dots, \alpha_4$.[4] Then let:

$$h_{4,\text{attn}} = \sum_{i=1}^{4} \alpha_i h_i$$

And now the word choice is $\text{argmax}\, Oh_{4,\text{attn}}$. Presumably α_1 will be large (if we've learned properly) and so 'Meowser' will better inform the completion of the sentence.

As we will see, since 2017, Transformer networks have gone a long way to replacing RNNs but use a lot of the techniques that RNNs incorporated to solve these problems.

[3] There are a lot of ways to do this. This is one such way.

[4] You may ask 'why convert into a_i at all?' This is left as an exercise to the reader.