**RNN, LSTM and GRU Networks**

[[Ellie Arbab](https://medium.com/@ellie.arbab?source=post_page-----e99bb056b571--------------------------------)](https://medium.com/@ellie.arbab?source=post_page-----e99bb056b571--------------------------------)

[Ellie Arbab](https://medium.com/@ellie.arbab?source=post_page-----e99bb056b571--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F32c26bf626d0&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40ellie.arbab%2Frnn-and-lstm-networks-e99bb056b571&user=Ellie+Arbab&userId=32c26bf626d0&source=post_page-32c26bf626d0----e99bb056b571---------------------post_header-----------)

5 min read

·

Nov 30

7

with an application in NLP text generation.

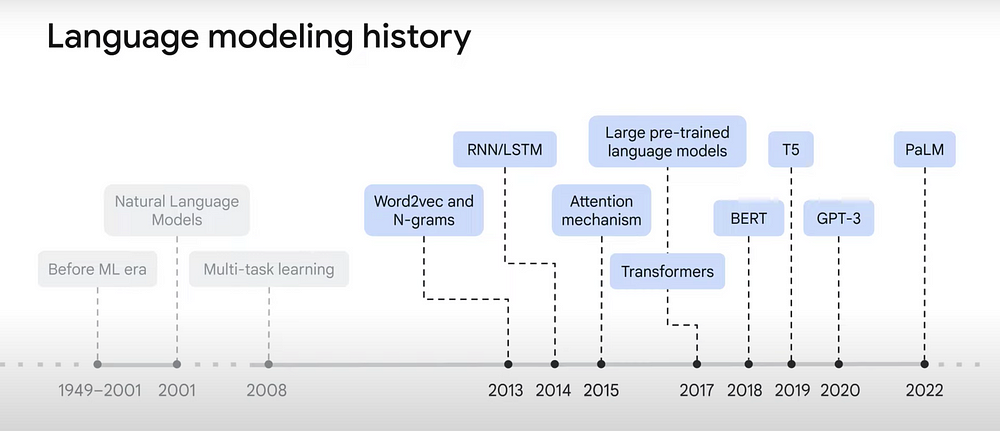


Figure 1: Progression of Language Models [1]

In previous posts we reviewed [Word2Vec](https://medium.com/@ellie.arbab/word-to-what-eb5a521e1f02), [Doc2Vec](https://medium.com/@ellie.arbab/doc-to-what-62c8139ffcaf), [GloVE](https://medium.com/@ellie.arbab/glove-8849a40c08bc), with a deep dive into the [Mathematics of Word2Vec](https://medium.com/@ellie.arbab/maths-of-word-to-vec-8af5d9c263f2). As the timeline in Figure 1 suggests, RNN and LSTM are next.

Similar to previous posts, we start with reviewing the architecture with all the low-level mathematics involved. We conclude the article with an application of LSTM’s in generating text.

**Recurrent Neural Networks (RNN)**

Recurrent Neural Networks (RNN) are a class of neural nets designed to model sequence events. It was first introduced in 1985 [2] and has since accumulated over 31,000 citations, according to scholar.google.com.

RNN’s architecture allow for making inferences on the next probable outcome based on what’s been observed thus far in a sequence of events.

Each layer *l*, has two activation functions, *a\_{lr}* and *a\_{ly}* which can be any of the familiar activations as fits, e.g. Sigmoid, Tanh, ReLU, etc. Recursion element, *r\_l*, embeds all information from the prior layers, *r\_*{*l-1*} and current layer’s input, *x\_l*. Each layer’s output, *y\_l*, in calculated based on *r\_l*, and so includes information from the current and previous layers.

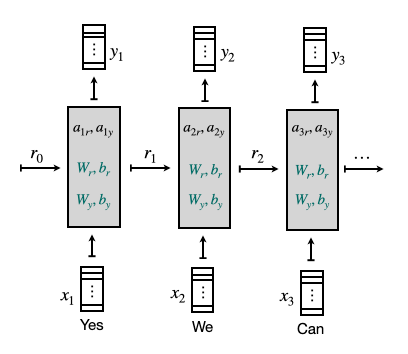
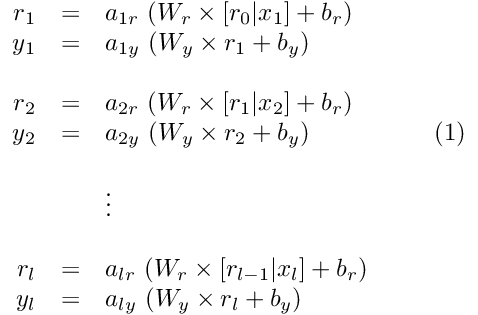


Figure 2: RNN Architecture: Recurrence, Activation, Weights and Biases.

Equation (1) describes the calculations inside the gray boxes of Figure (2).



Note that weights and biases, *W\_r*, *W\_y*, *b\_r, b\_y,* are shared in the entire network, noted in green in Figure (2), whereas activations, *a\_{lr}*, *a\_{ly}*, may vary at each layer.

This concludes the technical architecture of a standard RNN. But of course there’s more to it.

*For example, a****Bidirectional RNN****is one in which weights and biases are learned not only based on preceding inputs but upcoming ones as well. It is especially helpful when forthcoming events/words are crucial in building the correct or meaningful context for each layers’ and final outcome. Although Bidirectional RNNs sound more powerful than standard RNNs, when dealing with look-ahead-bias, e.g. in Time Series models, Bidirectional RNNs aren’t advisable.*

**Long Short-Term Memory Networks (LSTM)**

Long Short-Term Memory (LSTM) networks are a variation of RNN’s with the explicit memorization feature. It was first introduced in 1997 [3] and has since had over 88,000 citations, according to scholar.google.com.

LSTM’s follow a similar architecture as RNNs with an added vector *m,*for memory,whose job is to remember important nuances from previous layers, e.g. whether the subject in the sentence is singular or plural, tense of the previous sentence, etc. In the training phase, we learn *m’*s values as a combination of “update”, *u*, and “forget”, *f,* gates in addition to weights and biases for the output gate.

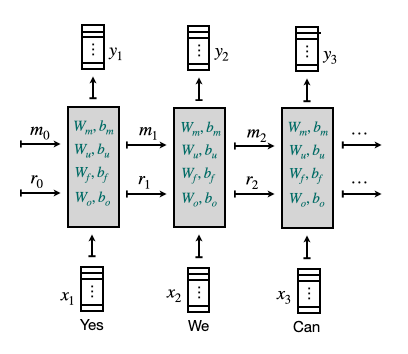
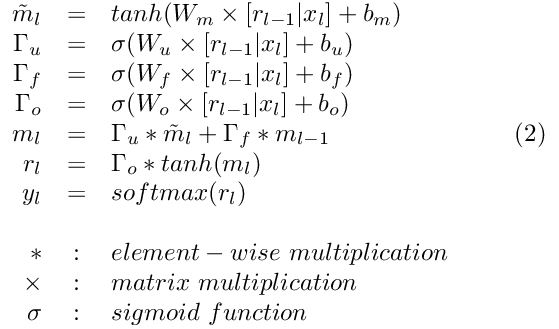


Figure 3: LSTM Architecture: Recurrence, Memory, Weights and Biases.

Equation (2) describes the calculations inside the gray boxes of Figure (3). Once again the items noted in green are shared in the entire network.



In LSTMs, at each timestamp we start with a temporary value for the memory cell, *m^tilde\_l*, based on previous timestamp’s recurrence value, *r\_{l-1},* and current input,*x\_l*. Previous recurrence, *r\_{l-1},* and current input, *x\_l,* are also used in learning the *update*, *forget* and *output* gates’ weights and biases, the *Γ*‘s.

Note how the final memory value is learned from a combination of update/forget gates.

Also note that in LSTMs, we settled the choice of activation functions to sigmoid and tanh compared to a standard RNN.

*Similar to RNNs and Bidirectional RNNs, LSTMs also come in a number of variations. For example,****Peephole LSTM****incorporates memory from the previous layer,*m\_{l-1}*, in calculating the gate values in addition to previous recurrence,*r\_{l-1}*, and current input,*x\_l*.*

LSTM’s are not the sole structure that model the concept of ‘memory’. In fact they are preceded by GRUs (Gated Recurrent Unit). In practice LSTM’s tend to outperform GRUs in various tasks. But now that we’ve covered LSTMs in full, understanding GRUs should be self-explanatory!

**Gated Recurrent Unit (GRU)**

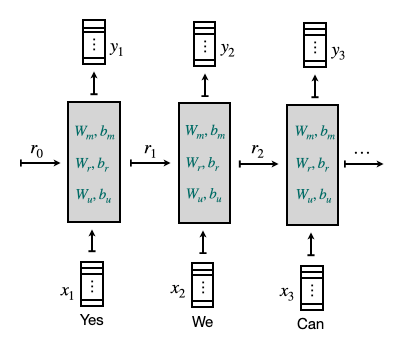
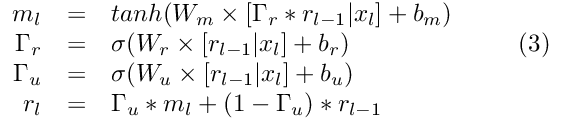


Figure 4: GRU Architecture: Recurrence, Memory, Weights and Biases.



**Application**

Let’s build an NLP application. The task is to train an RNN-LSTM to predict the next word given a sequence of seed words.

This code snippet shows how to set up an RNN with 2 LSTM layers, 2 Dropout layers and, one fully connected Dense layer. For training, I used all available US presidential inauguration speeches, on a rolling window of 10 words as input and the next word in sequence as output.

model = Sequential()  
embedding\_layer = Embedding(num\_words,   
 EMBEDDING\_DIM,   
 weights = [embedding\_matrix],   
 input\_length = seq\_len,   
 trainable = False)  
model.add(embedding\_layer)  
model.add(LSTM(256, input\_shape = (X.shape[1], X.shape[2]),   
 return\_sequences=True))  
model.add(Dropout(0.2))  
model.add(LSTM(256))  
model.add(Dropout(0.2))  
model.add(Dense(y.shape[1], activation="softmax"))  
model.compile(loss="categorical\_crossentropy", optimizer="adam")

*Prompting the model to predict the next 3 words given the following seed:  
“my fellow americans it is time to stand together and ”  
resulted in:  
“uncomplaining interpreted seize”*

I have to agree it doesn’t ensue the most prolific speech, although the predicted words are well in-tune with the seed’s sentiment. Given the only 58 sample speeches with 45 different literally styles to train on, one might say it’s in fact pretty good outcome!

You may find [the full code on Github](https://github.com/earbab/RNN_LSTM_BlogPost/blob/main/RNN_LSTM_demo.ipynb).

Recap  
RNNs are the neural networks that make inferences based on a sequence of inputs. They are widely used in NLP and TimeSeries applications. LSTMs are a special subset of RNNs where the network learns which segment of the past is still relevant and what can be forgotten.

We covered the standard RNN, LSTM and GRU architectures, which really is a blueprint for an infinite variation of each. It is also important to keep in mind that the robustness and performance of these standard architectures are well researched. So it’s advisable to make sure there is material value-add and that impact scope is well understood before tampering with the standard architecture.

**Reference**

[1] Google Cloud Tech, [Transformer Models and Bert Model: Overview](https://www.youtube.com/watch?v=t45S_MwAcOw).

[2] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. “[Learning internal representations by error propagation.](https://apps.dtic.mil/sti/pdfs/ADA164453.pdf)” (1985).

[3] Hochreiter, Sepp, and Jürgen Schmidhuber. “[Long short-term memory.](https://papers.baulab.info/Hochreiter-1997.pdf)” *Neural computation* 9, no. 8 (1997): 1735–1780.