**Resources**

**Concepts Explained Simply**

<https://distill.pub/>

**Visualizing Seq2Seq Models**

<https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>

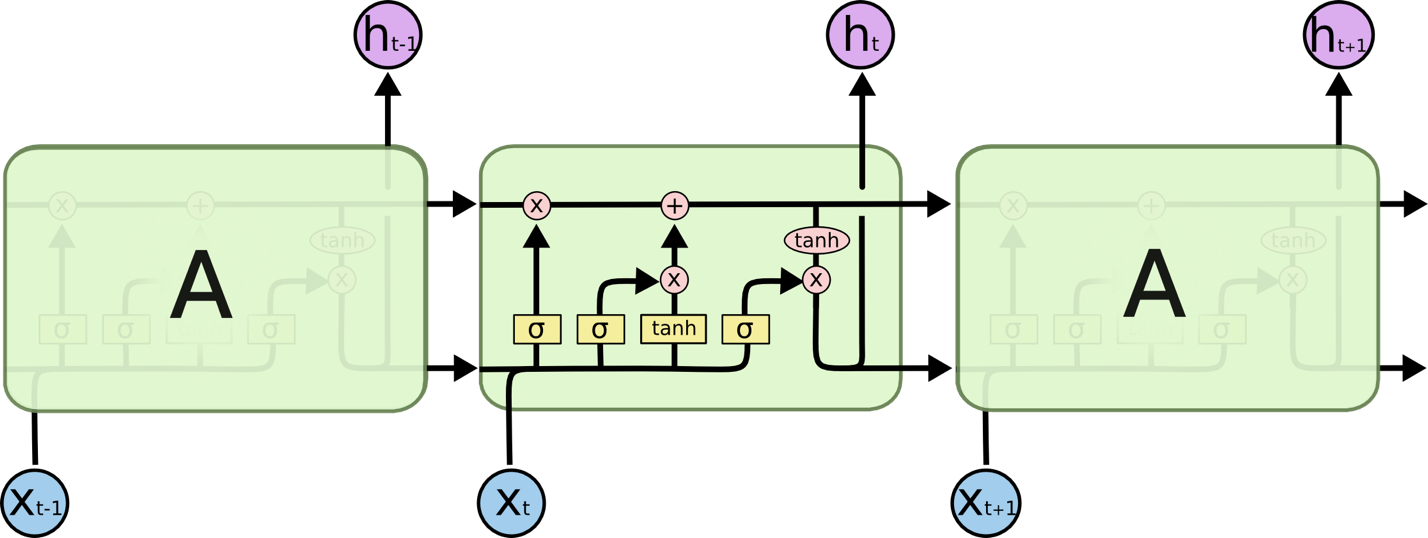
**Understanding LSTM Networks**

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

<https://www.youtube.com/watch?v=WCUNPb-5EYI>

<https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714>

<https://medium.com/@divyanshu132/lstm-and-its-equations-5ee9246d04af>



**Bert**

<https://www.youtube.com/watch?v=BhlOGGzC0Q0>

**Useful Blogs**

<http://colah.github.io/>

<https://e2eml.school/blog.html>

**Important Functions/Points**

* When training RNNs – each feature will be the sequence of things before it
  + For words – each feature would be (for example) the length of the 50 words and the label would be the 51st word
    - This means you’d get a training data of the shape (num\_samples, 50, 1)
    - This is because in the language of recurrent neural networks – each sequence **has 50 timesteps, each with 1 feature**
* When given an array which has not been properly formatted in terms of columns e.g. [1 2 3 4 5 6] – set array = array.reshape(-1, 1)
  + Use of this function in numpy is to properly recognize/format the array into the correct dimension of [1, 2, 3…etc.]
* Can also apply a transform to transform all numerical data between 0 and 1
  + To do this in scikit learn:
    - from sklearn.preprocessing import MinMaxScaler
    - Sc = MinMaxScaler(feature\_range=(0,1))
    - Training\_set\_scaled = sc.fit\_transform(training\_set)
  + \*\* DON’T FORGET TO REVERSE TRANSFORM AFTER YOU GENERATE YOUR PREDICTIONS
    - Predictions = sc.inverse\_transform(predicions)
* Neural networks will work better if your labels are one-hot encoded
  + In general when thinking about sizes – a label needs to have as many rows as the features and as many columns as there are possibilities. For example, if the feature was of shape (2000, 1) and there were 4 possibilities for output – shape of the labels will be (2000, 4) where the each label is of type [0, 0, 0, 0]
* Can quickly produce simple plots with df.plot(x=.., y =..)
* Given a one-hot encoded array or even one that is not np.argmax() is a useful function for extracting the actual label
  + Np.argmax([1,2,3,2,1]) would return 2 since this array implies a label that looks like [0,0,1,0,0] and thus corresponds to whatever the 3rd item in this array represents
* Given a numpy array like a=np.array([[2, 3, 1, 0], [1,2,3,4], [2,3,4,5]])
  + a[:,0] will return an array with the elements in all of the 1st column i.e. [2, 1, 2]

LSTM Models

* If we’re building an LSTM network that uses more than one layer of LSTM cells i.e. involves stacking two or more LSTM layers, we MUST return the sequences (which would be passed into the LSTM layers)
* Dropout should be added to avoid overfitting
* **Input to an LSTM layer always has the** (batch\_size, timesteps, features) shape
  + For the tesla stock example, since we were only using high, features was 1. But imagine if we were doing some kind of text classification problem where we converted words into vectors – features would be dimension of those vectors

Keep Up W. Ongoing Research

<http://arxiv-sanity.com/>

**Research Notes**

<https://github.com/JasonBenn/deep-learning-paper-notes>

**Short Crash Course on RNNs**

* An RNN takes two inputs: the actual input vector and the hidden state
* It produces two outputs: the output vector and the new hidden state (of the data that was just processed)