ACENET
accelerate discovery

Recurrent Neural Networks

with
Daniel L. Silver, Ph.D.
Andy McIntyre, Ph.D.

ACADIA INSTITUTE FOR DATA ANALYTICS
digital harvest

June 24, 2022

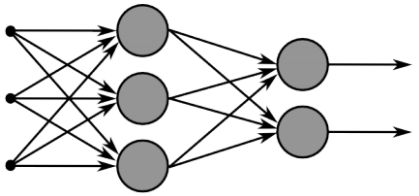
DEMO:

Aficionado

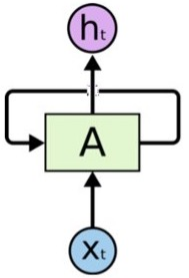
Slides based on originals by Yann LeCun

2022-06-24

Recurrent Neural Networks



- Multi-layer feed-forward NN: **DAG**
  - Just computes a fixed sequence of non-linear learned transformations to convert an input pattern into an output pattern

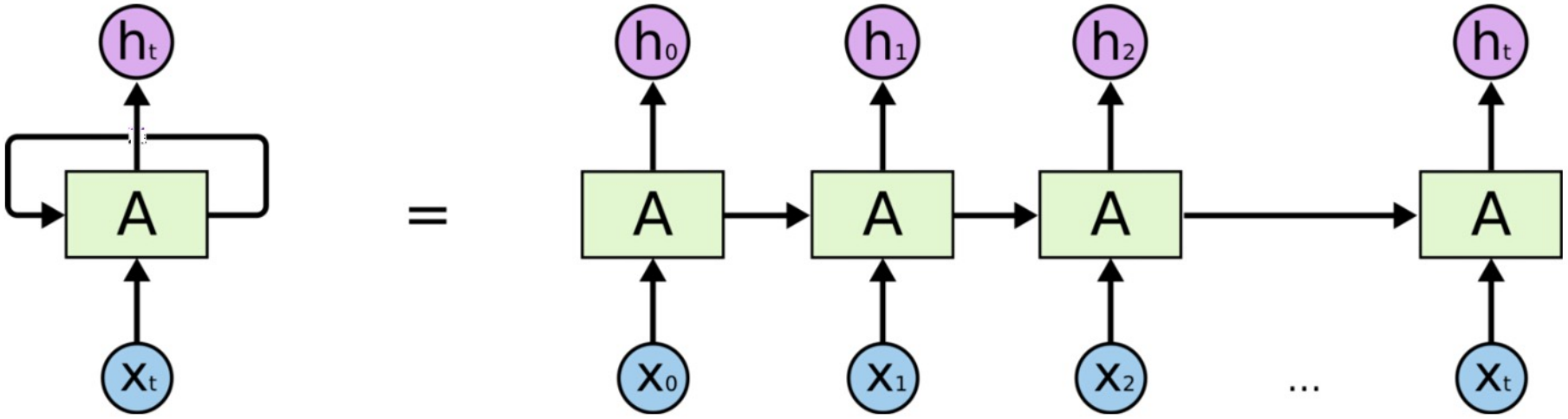


- Recurrent Neural Network: **Digraph**
  - Has a cycle of information
  - Cycle can act as a memory (or context)
  - The hidden state of a recurrent net can carry along information about a "potentially" unbounded number of previous inputs.
  - Can model sequential data in a much more natural way.

Simple Recurrent Neural Networks

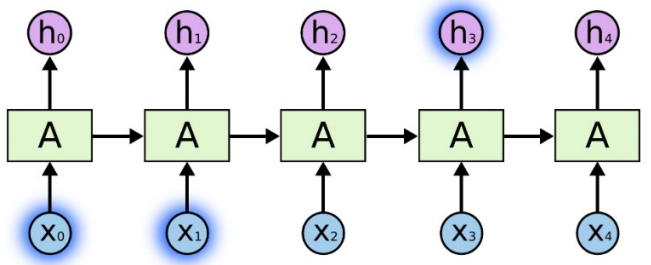
$$h_t = f_W(h_{t-1}, x_t)$$

new state — some function with parameters W — old state — input vector at some time step



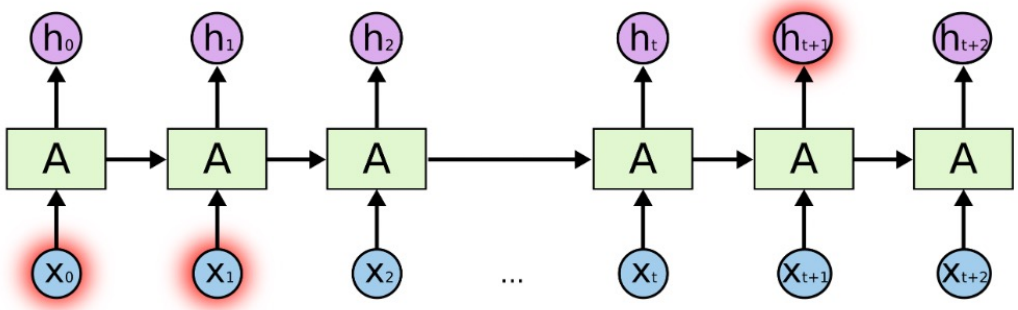
Short-term Memory in RNN

- The context units remember the previous internal state.
- Thus, the hidden units have the task of mapping both an external input and also the previous internal state to some desired output.
- An RNN can predict the next item in a sequence from the current and preceding input.
- Great for learning a song!



Basic RNN Limitations

- Vanishing/Exploding Gradients
  - RNN can be considered a deep ANN
  - Back Prop Through Time (BPTT)
  - Gradient can become very **small or very large quickly**, and the locality assumption of gradient descent breaks down (Vanishing gradient) [Bengio et al 1994]
- Results in a limited (short-term) memory
  - as that gap grows, RNNs become unable to learn to connect the information

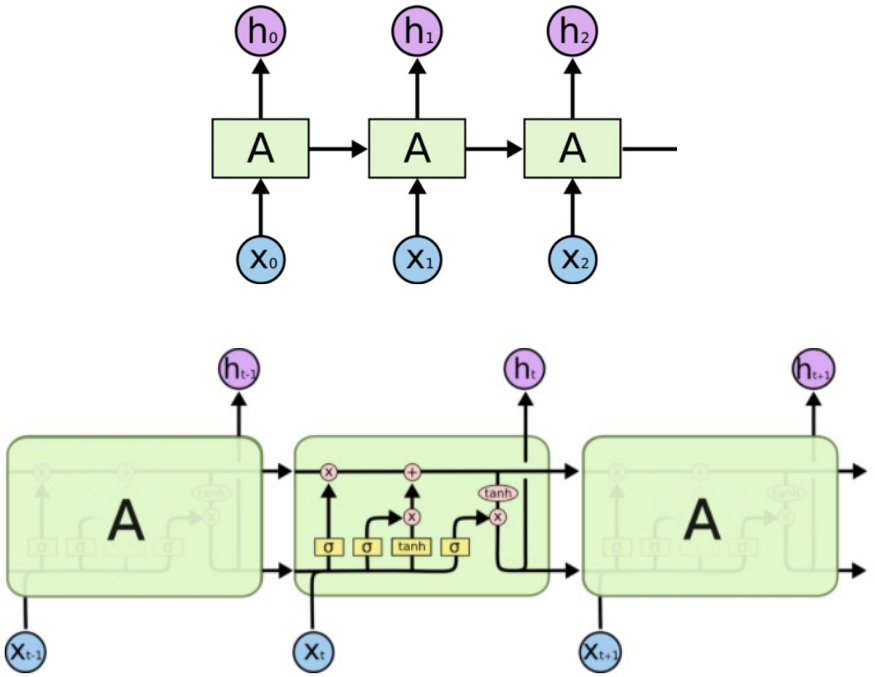


Vanishing/Exploding gradients

- In an RNN trained on long sequences (*e.g.* 100 time steps) the gradients can easily explode or vanish.
  - So RNNs have difficulty with long-range dependencies.
- Tricks to handle gradient problems:
  - Clip gradients with bigger sizes (exploding gradient)
  - Use dropout to regularize
  - Recurrent input normalization
- Many methods proposed for reduce the effect of vanishing gradients; although it is still a problem
  - Abandon stochastic gradient descent in favor of a much more sophisticated Hessian-Free (HF) optimization
  - Add fancier modules that are robust to handling long memory; e.g. **Long Short Term Memory (LSTM)**

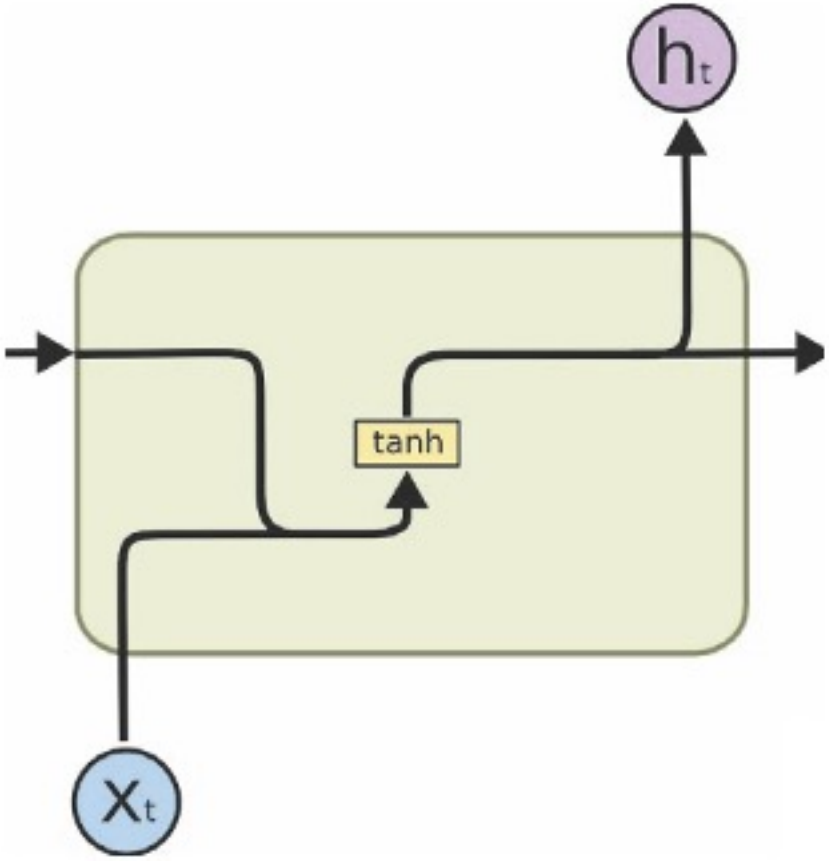
Long Short Term Memory (LSTM)

- Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps).
- They designed a memory cell using logistic and linear units with multiplicative interactions
- Key idea is that information from the distant past can be retained if important to model

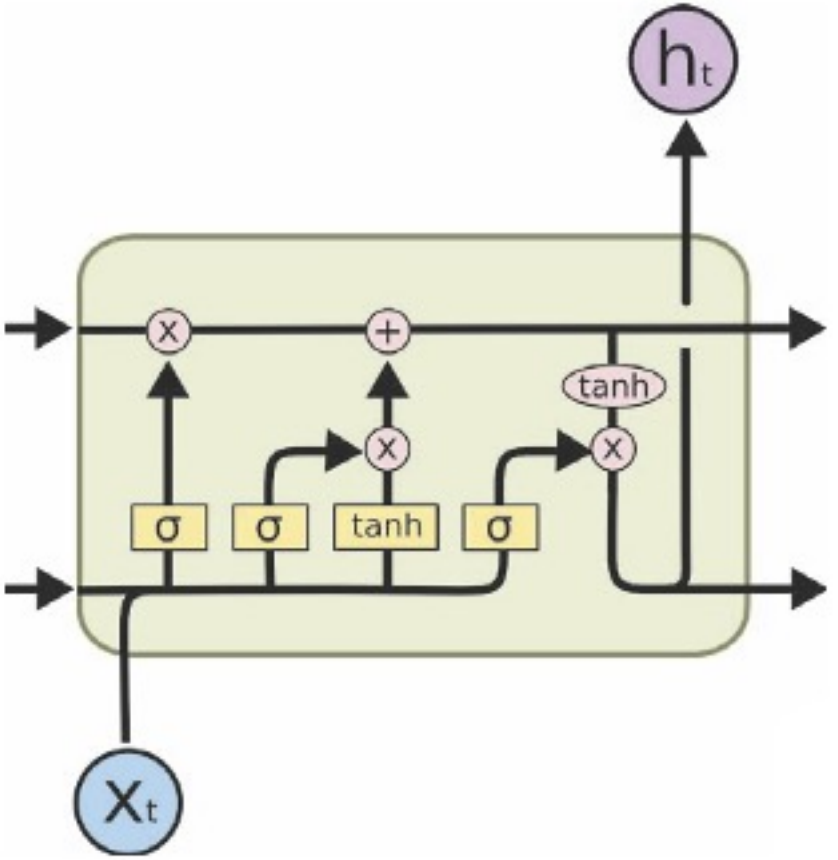


The repeating module in an LSTM contains four interacting layers.

RNN vs LSTM

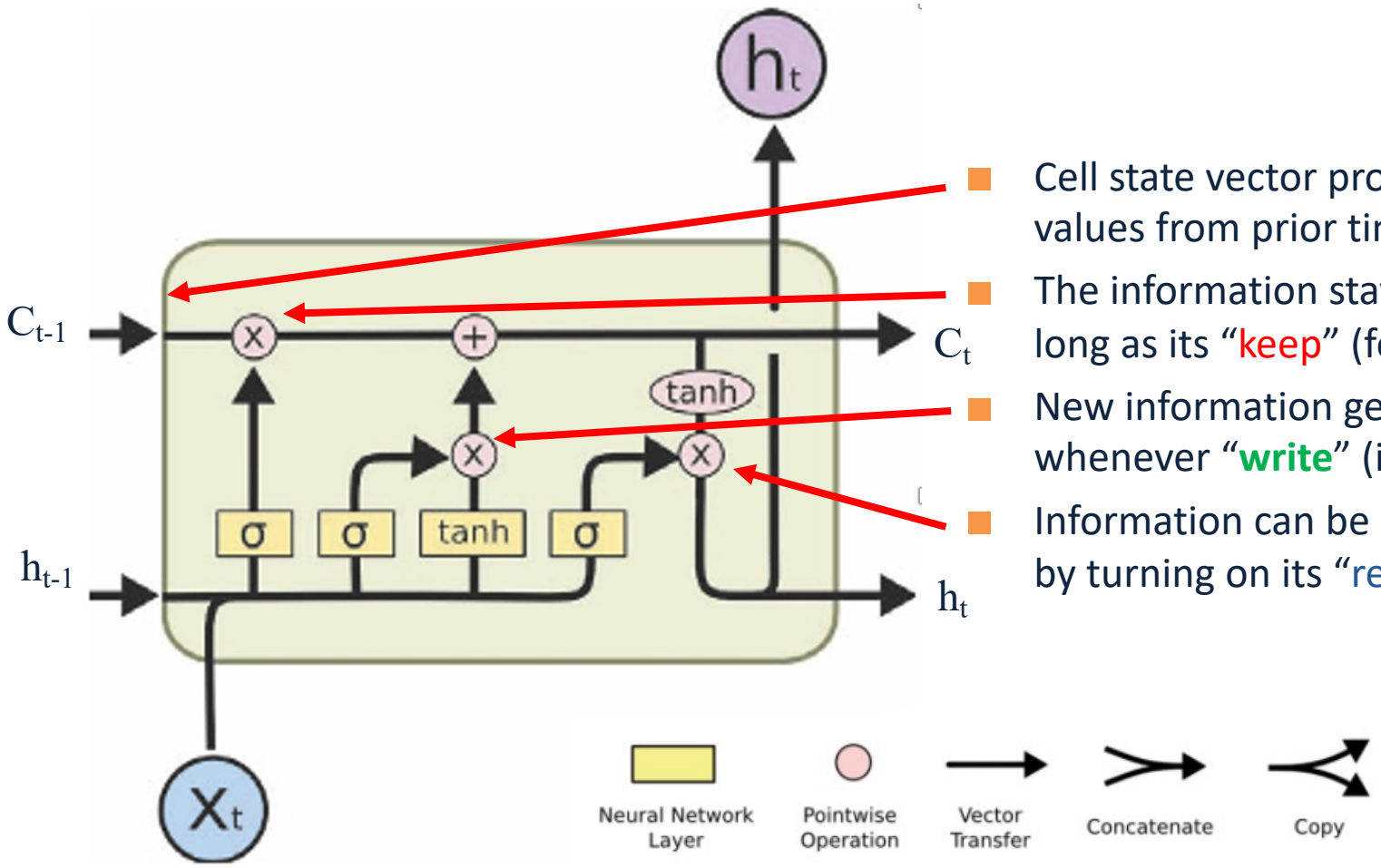


(a) RNN



(b) LSTM

Long Short Term Memory (LSTM)



- Cell state vector provides memory of values from prior time steps
- The information stays in the cell so long as its "keep" (forget) gate is on.
- New information gets into the cell whenever "**write**" (input) gate is on.
- Information can be read from the cell by turning on its "read" gate.

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

Keras-Tensorflow return_sequence

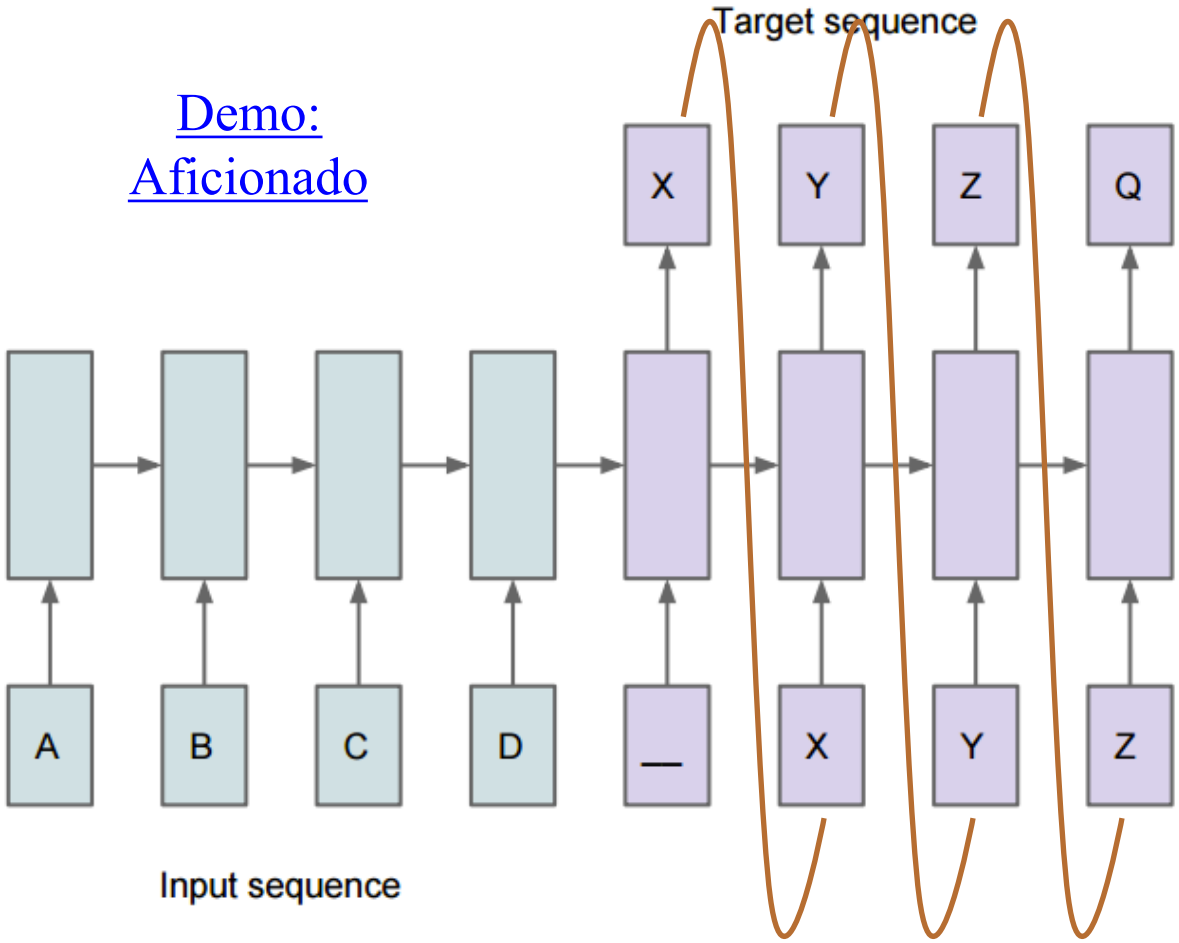
- By default an RNN layer has **return_sequence=False**
- This means the layer will return a 2D array containing only the output for last time step

- If you set RNN layer to **return_sequence=True**
- This means that it will return a 3D array containing an output for all time steps

- https://machinelearningmastery.com/return-sequences-and-return-states-for-lstms-in-keras/

LSTM RNNs – can learn a long song

Demo: Aficionado



Random Seeds in Tensorflow/Keras

What do you think the output of this should be?

```
a = tf.random.shuffle([1,2,3,4,5,6], seed=567436).numpy()
b = tf.random.shuffle([1,2,3,4,5,6], seed=567436).numpy()
a,b
```

```
(array([4, 2, 6, 1, 5, 3], dtype=int32), array([1, 4, 2, 5, 6, 3], dtype=int32))
```

What you should be doing is:

```
tf.random.set_seed(99)
a = tf.random.shuffle([1,2,3,4,5,6], seed=567436).numpy()
tf.random.set_seed(99)
b = tf.random.shuffle([1,2,3,4,5,6], seed=567436).numpy()
a,b
```

```
(array([6, 4, 2, 1, 5, 3], dtype=int32), array([6, 4, 2, 1, 5, 3], dtype=int32))
```

```
import random
import numpy as np
import tensorflow as tf
random.seed(seed)
np.random.seed(seed)
tf.random.set_seed(seed)
```

To set all random seeds for Python, NumPy, and TensorFlow:

```
tf.keras.utils.set_random_seed(99)
```

https://www.tensorflow.org/api_docs/python/tf/keras/utils/set_random_seed

TUTORIAL 9

- Develop and train a LSTM network using Keras and Tensorflow
  - tf.keras_rnn.ipynb

- For **much** more (text generation, time series prediction):
  - https://www.tensorflow.org/tutorials/sequences/text_generation
  - https://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/

ADDITIONAL TUTORIALS

1. https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/
2. https://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/
3. https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
4. https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/
5. https://www.tensorflow.org/tutorials/sequences/text_generation

6. Specific to CNN LSTM
   - https://machinelearningmastery.com/cnn-long-short-term-memory-networks/
   - https://towardsdatascience.com/get-started-with-using-cnn-lstm-for-forecasting-6f0f4dde5826

References

- Christopher Olah’s Blog
  - http://colah.github.io/
  - http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Juergen Schmidhuber's tutorial
  - http://people.idsia.ch/~juergen/lstm/
- Assaad Moawad's blog
  - https://medium.com/datathings/the-magic-of-lstm-neural-networks-6775e8b540cd