

LSTMs, with Musical Demo

Joel Gitter & Kindeep Singh Kargil

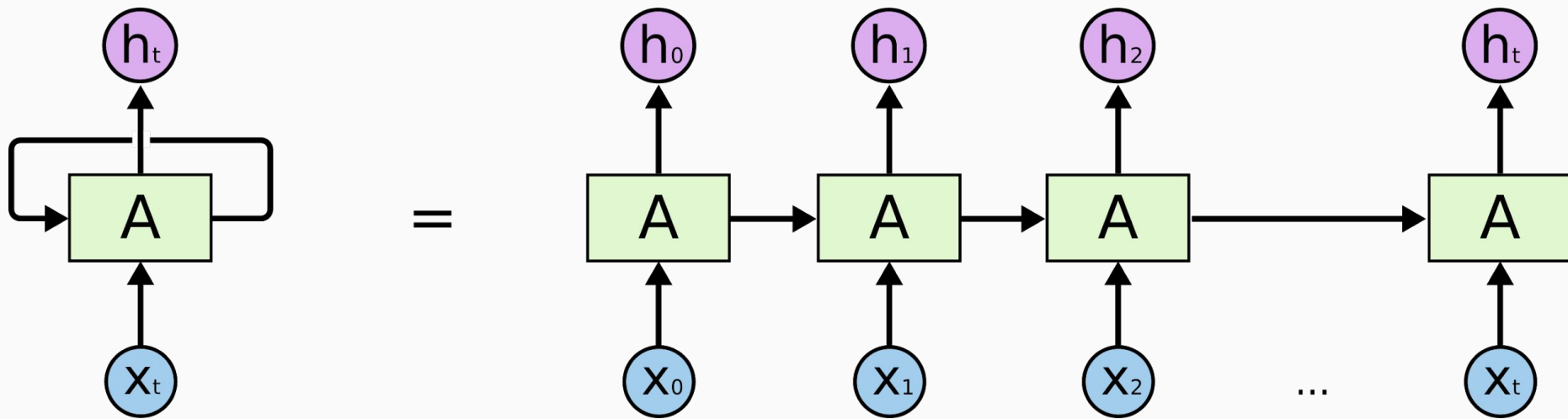
COSC 4P80

March 29, 2021

Part 1: The Info

What are RNNs?

- RNN = “Recurrent Neural Network”
- In essence, neural networks that pass output from the current iteration to the next iteration
- This implies that (output vector size == input vector size)
- Particularly useful for datasets that have context involved, like time-series or sequential data



The Vanishing Gradient Problem

- Problem specifically related to backpropagation of recurrent neural networks
- As the backpropagation makes corrections further and further back, the correction gradient exponentially approaches 0, thus “vanishing”
- Mathematically, this is caused by the one vector’s derivative being calculated with respect to another vector [4]
- Opposite of the “exploding gradient” problem

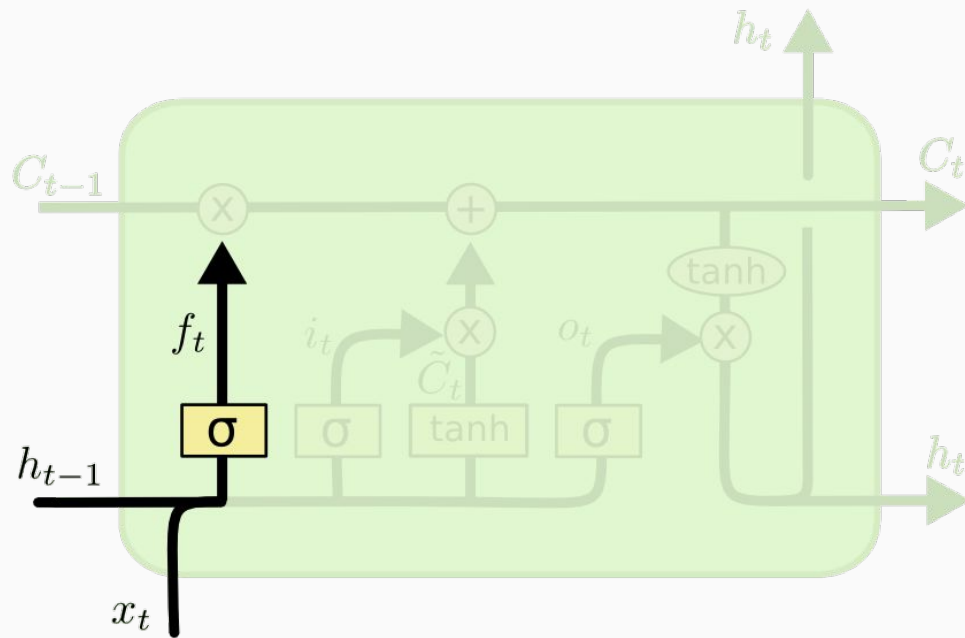
What are LSTMs?

- LSTM = “Long Short Term Memory”
- RNN variant, solves the vanishing gradient problem
- Effectively 4 NNs combined into one:
 - Predict - takes in the new information, makes a prediction
 - Ignore - takes in the historical information, chooses to ignore current prediction(s)
 - Forget - takes in historical information, chooses to “forget” previous predictions(s)
 - Select - takes in historical information, filters current prediction(s)

How LSTMs Work

Cell State: Forget gate

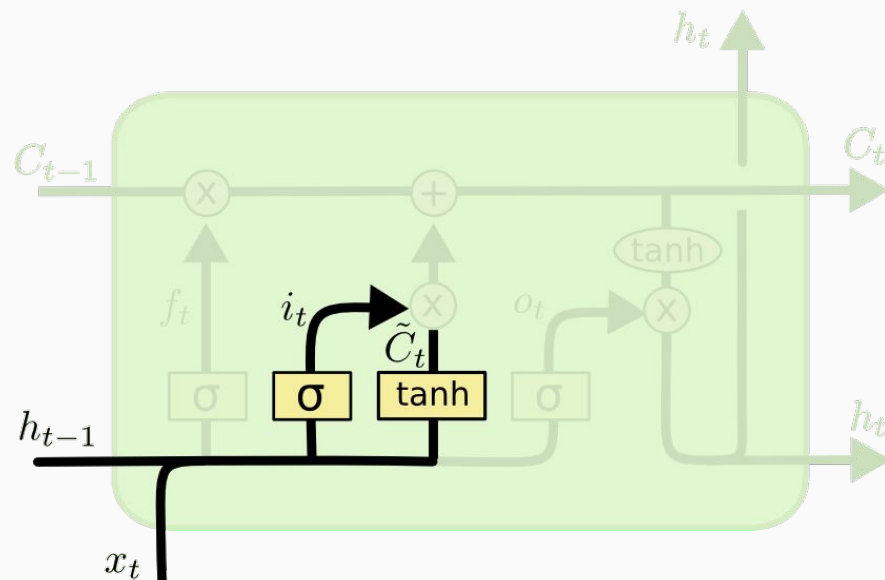
- Use output from previous LSTM cell and new input
- Decide what information to keep and what to do away with
- Forget vector: components between 0 and 1. Decides how much of each component to keep, 0 means nothing and 1 means keep it as is.
- Cell state C is multiplied by forget vector



How LSTMs Work

Cell state: Determine update

- Input gate layer: Determine what should be updated, a vector of values between 0s and 1s using sigmoid
- tanh layer: new candidate values
- Combine to get update



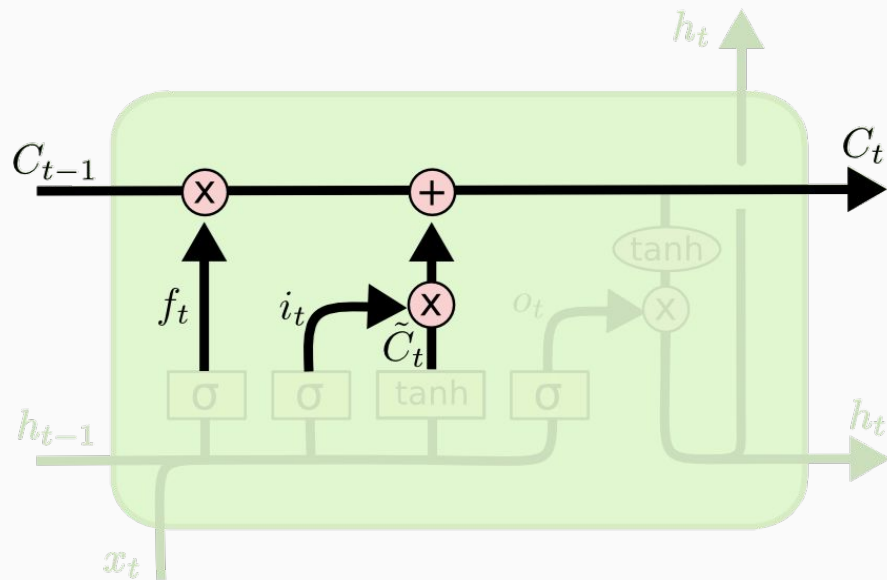
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

How LSTMs Work

Cell state: determine update

- Combine and add to cell state after the forget layer to get new cell state



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

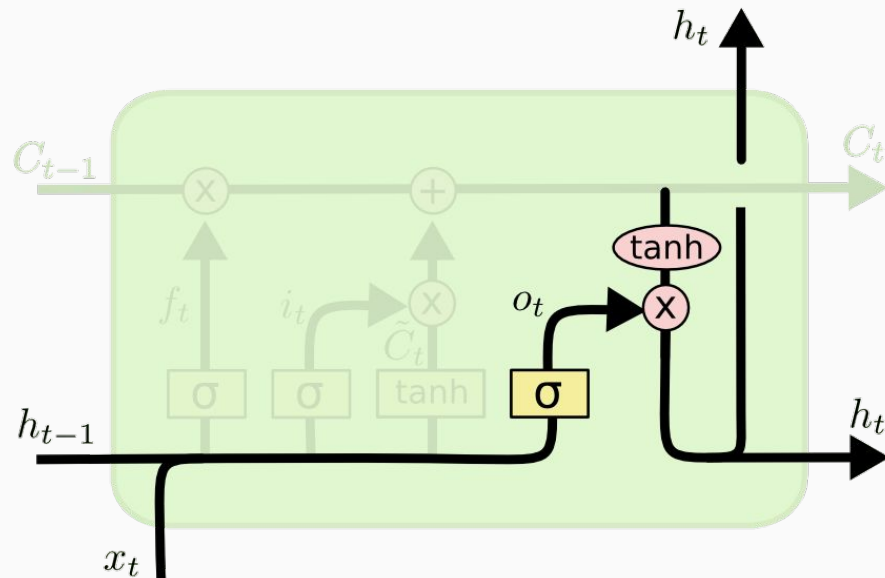
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

How LSTMs Work

Output

- Sigmoid layer to decide what to update from our combined input
- Cell state influence: Tanh on cell state
- Multiply to get output

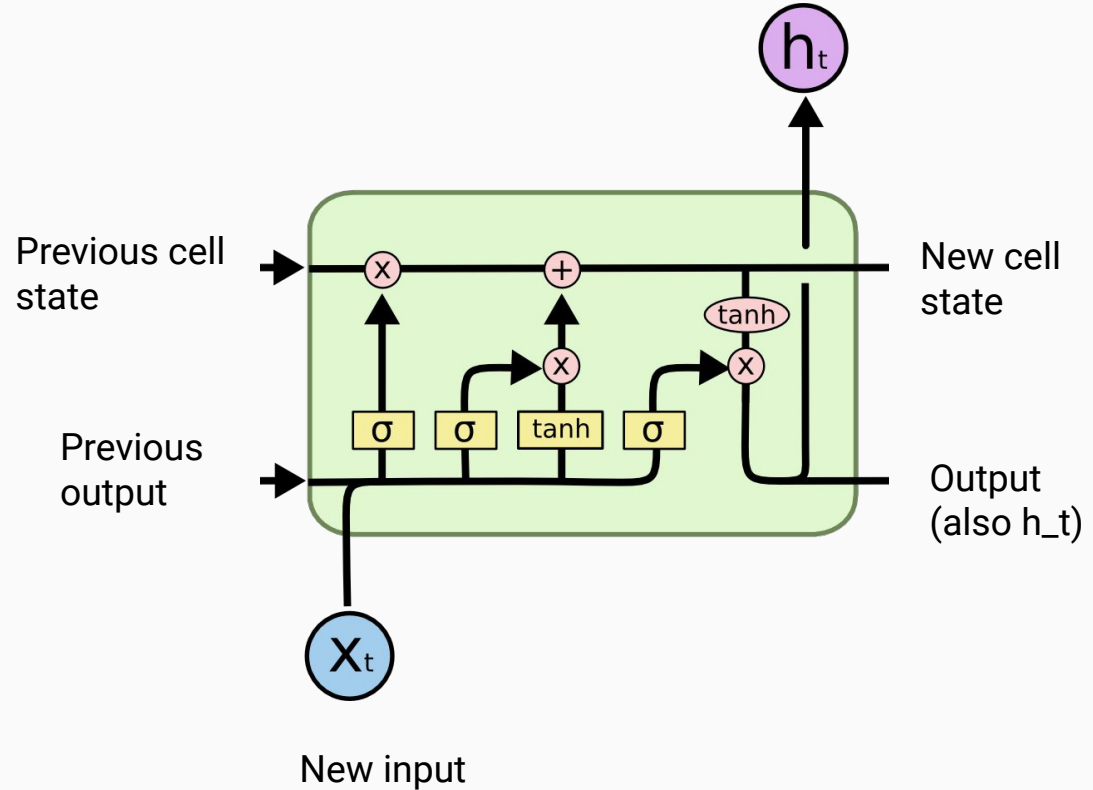


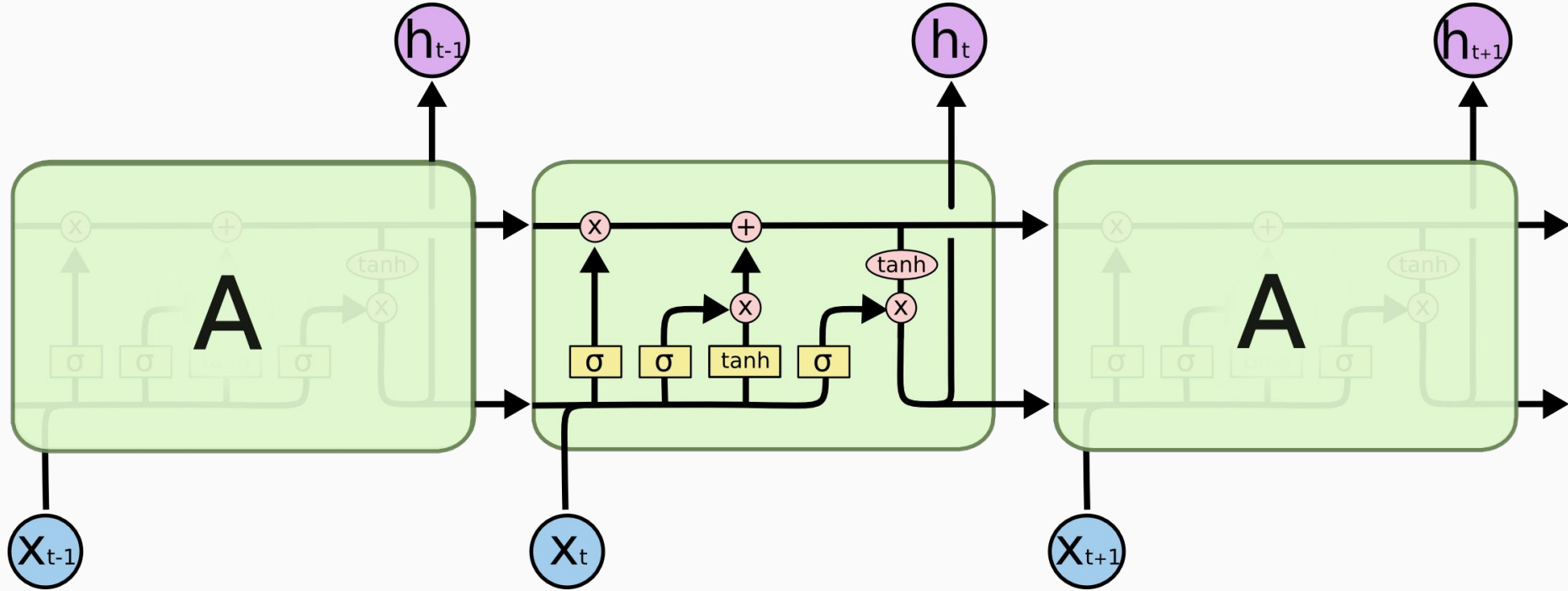
$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

Summary

- Determine what to keep from previous cell state
- Update cell state
- Use new cell state to determine output

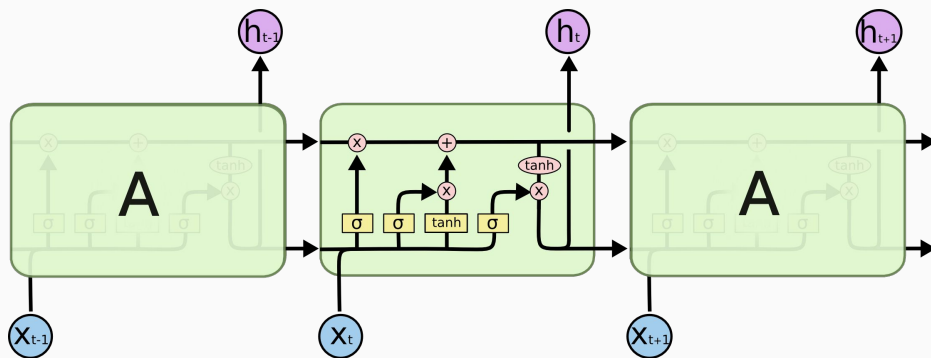




Graphical representation of a LSTM - from [3]

Training an LSTM

- It's also backprop, propagate from time step t back to 0
- Backpropagation Through Time (BPTT)



Common Use-cases of LSTMs

- NLP
- Speech recognition
- Language Translation
- Time series prediction
- Market prediction

Part 2: The Demo

The Demo: Predicting the Next Note







- The concept: Given the notes preceding, can we build a LSTM model to predict the next note?
- The data: Sonata No. 10 in C Major, K. 330, by Mozart
- The platform: Keras, using Google Colab
- Keras model setup:
 - Using an LSTM layer, followed by a Dense layer
 - Loss function: `categorical_crossentropy`
 - Optimizer: ADAM

Screenshot: Colab notebook

The Demo: Data Pre-processing

- Notes were manually transcribed from the sheet music to a notation indicating the key and an octave modifier
 - E.g. C#++ indicates C sharp, two octaves above middle C
- Once in this note notation, notes can be transformed into an 88-ary array indicating which note is currently being played
- For X (input): take the previous 20 88-ary arrays
- For Y (result): take the current 88-ary array
- Train_test_split: test_size=0.2, shuffle=True

The Demo: Actual vs. Predicted Next Note

	Prior Notes	Actual Next Note	Predicted Next Note
Correct			
Incorrect			

72.31%

Accuracy of predicted note matching the actual note

Part 3: The Conclusion

References

Resources used include:

[1] <https://www.bioinf.jku.at/publications/older/2604.pdf>

[2] <http://www.bioinf.jku.at/publications/older/2304.pdf>

[3] <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

[4] https://compstat-lmu.github.io/seminar_nlp_ss20/recurrent-neural-networks-and-their-applications-in-nlp.html

[5] <https://youtu.be/WCUNPb-5EYI>

Summary, and Questions?

Summary of what we've covered:

- LSTMs = RNNs + selective memory
- LSTMs were designed to solve the vanishing gradient problem
- LSTMs, like RNNs, are particularly useful when context is involved, such as time-series or sequential data
- Built an experiment to predict the next note in a song, with 72.31% accuracy

Try for yourself!

<https://github.com/JoelGritter/LSTM-predict-note>

