Spyros Chatzivasileiadis and Haris Ziras

46040 Introduction to Energy Analytics

Reviewing Previous Lecture

- For 5 minutes discuss with the person sitting next to you about:
  - Three main points we discussed in the last lecture
  - One topic or concept that is not so clear and you would like to hear again about it

Hands-on of March 20

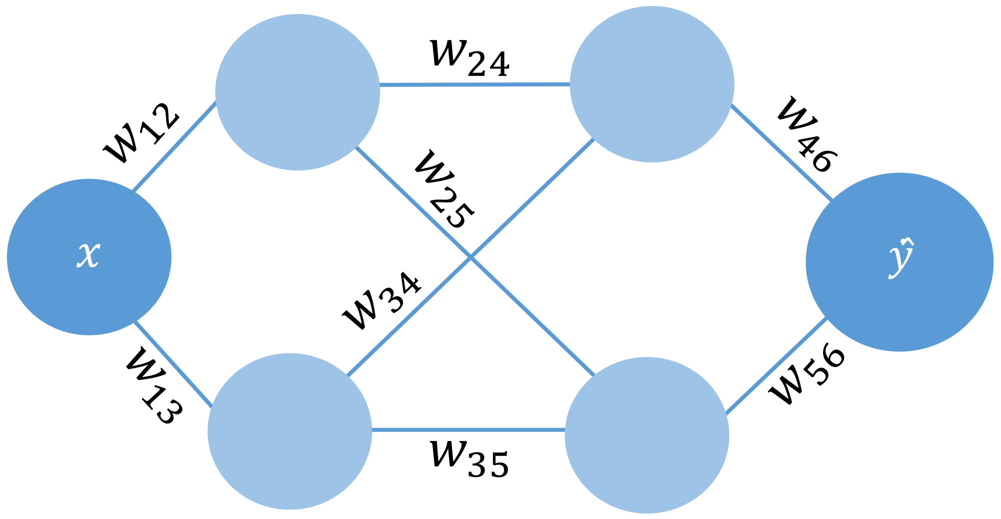
- What are the main takeaways of the hands-on for the ones who tried it?

- How does the NN structure look like?
- What is the activation function used?

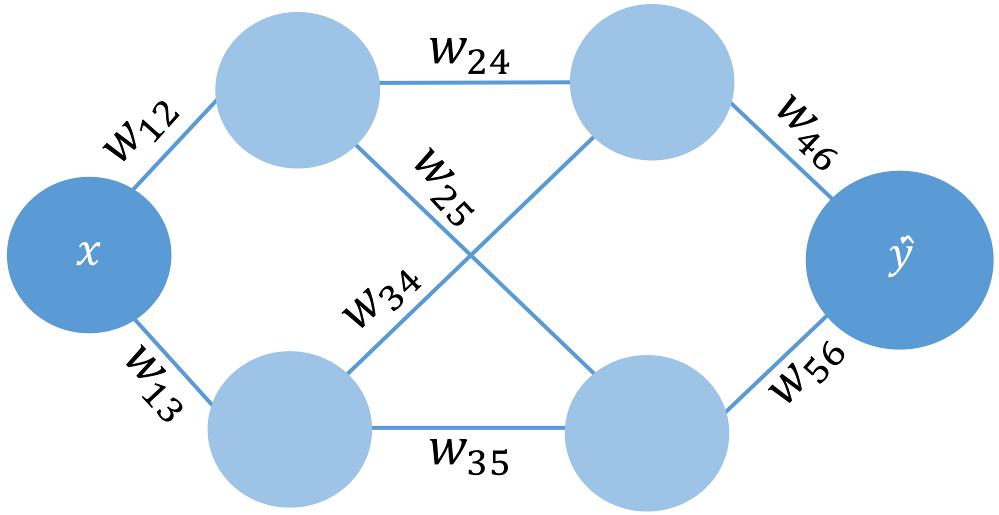
Packaged you will need for the assignment

- Pandas
- Tensorflow
- Matplotlib
- Keras
- Scikit_learn
- Numpy = 1.18.5
- Seaborn

What is the challenge of Neural Networks when it comes to time-series applications?

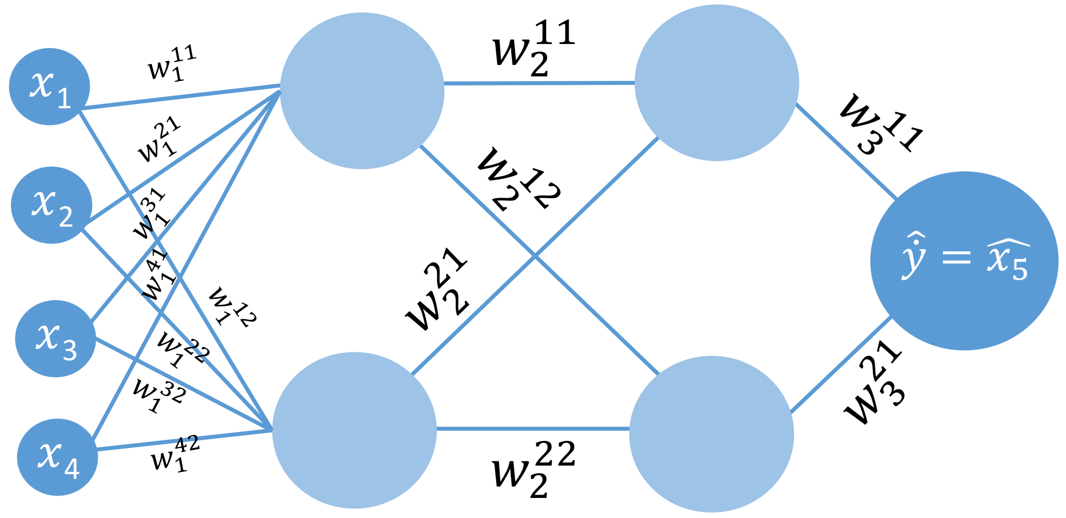


What is the challenge of Neural Networks when it comes to time-series applications?



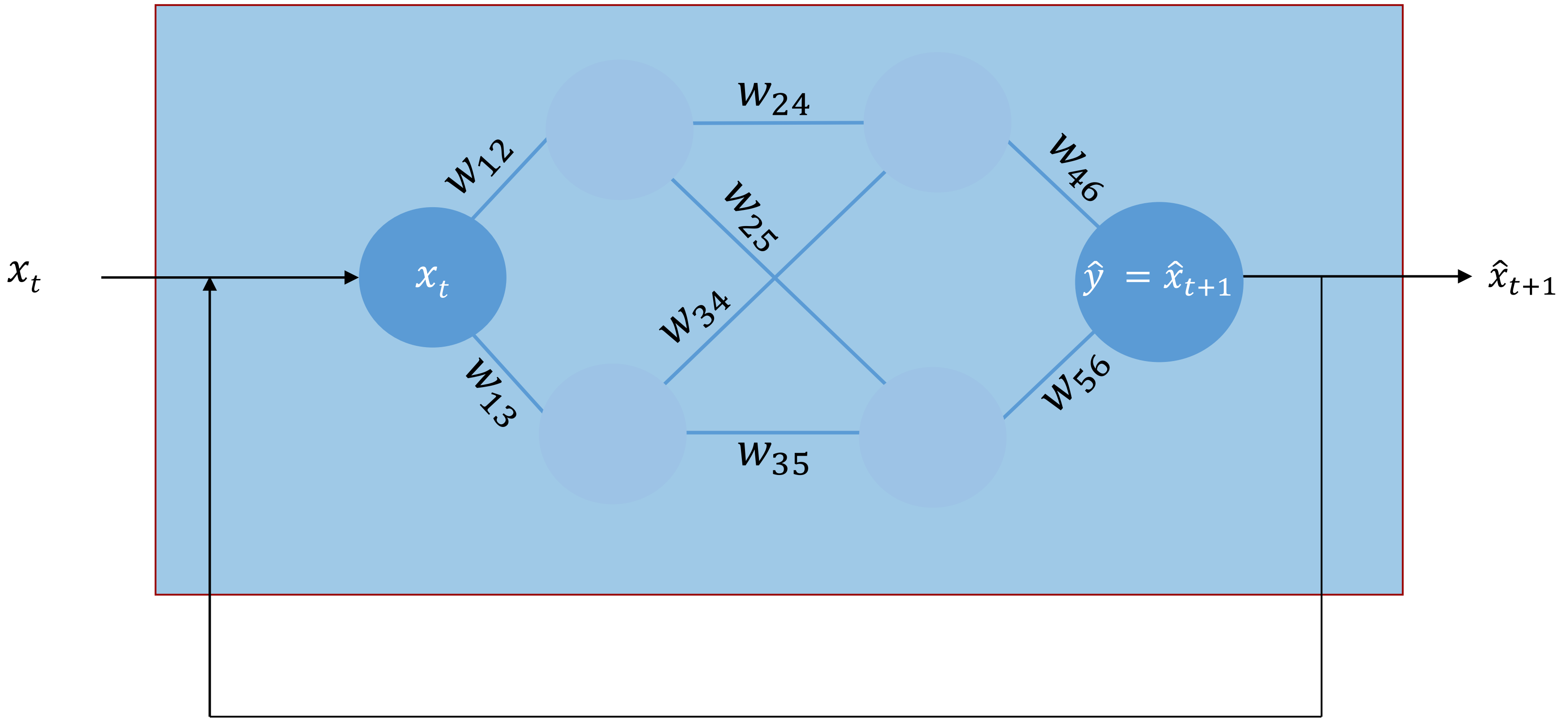
- No memory!
- Every new input x is completely new information for them
  - The output $\hat{y}$ is not dependent on previous inputs or outputs
- Time Series forecasting (e.g. prices, temperature, wind, etc.) depends on previous values → requires memory
  - And so does natural language processing, and many others!

Could I add some sort of "memory" to the standard Neural Networks? Yes, but not efficiently

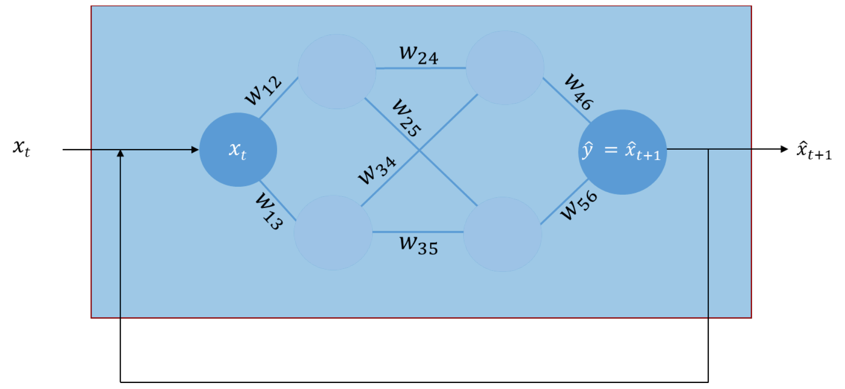


- We can create a time series input. For example, we can enter the previous 4 values in order to predict $\hat{x}_5$
- In that way, every output will be predicted based on the previous 4 inputs.
- However, are 4 inputs enough?
- And what if my next output is dependent on the previous 1000 inputs?

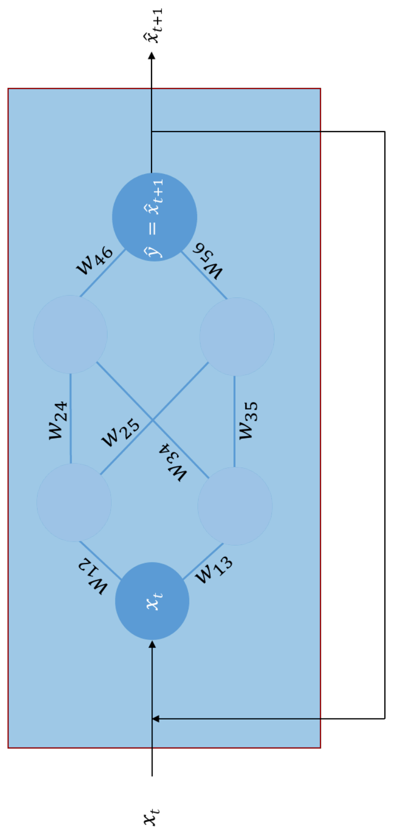
Recurrent Neural Network



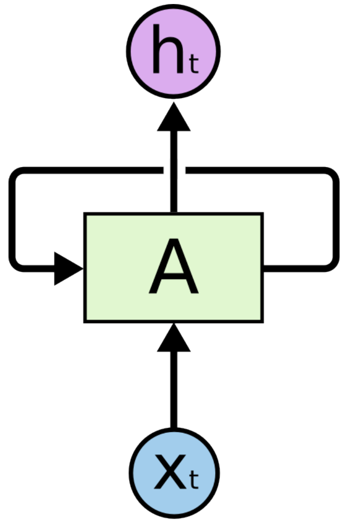
Recurrent Neural Network (just introducing a different graphical representation which we will use for the rest of this lecture)



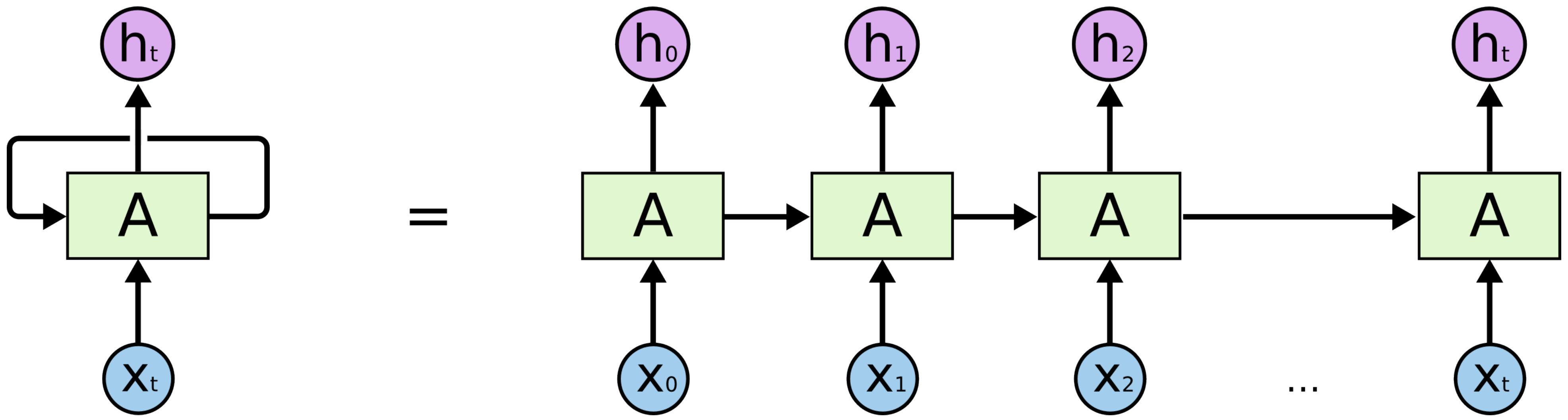
=



=



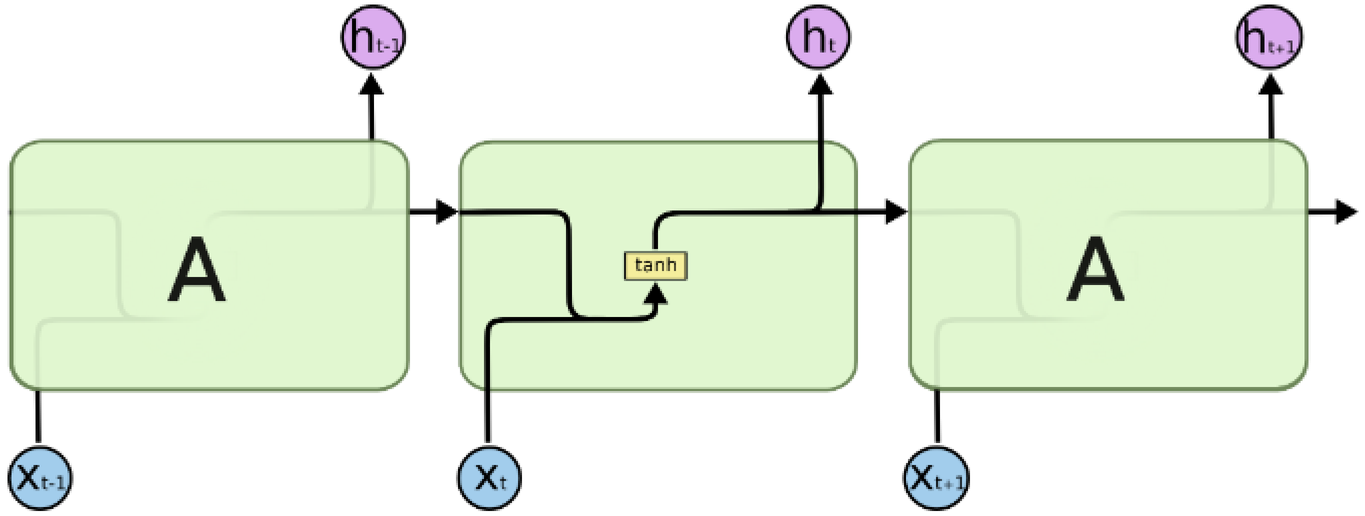
Recurrent Neural Networks



- A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.
- Here is what happens if we unroll the loop

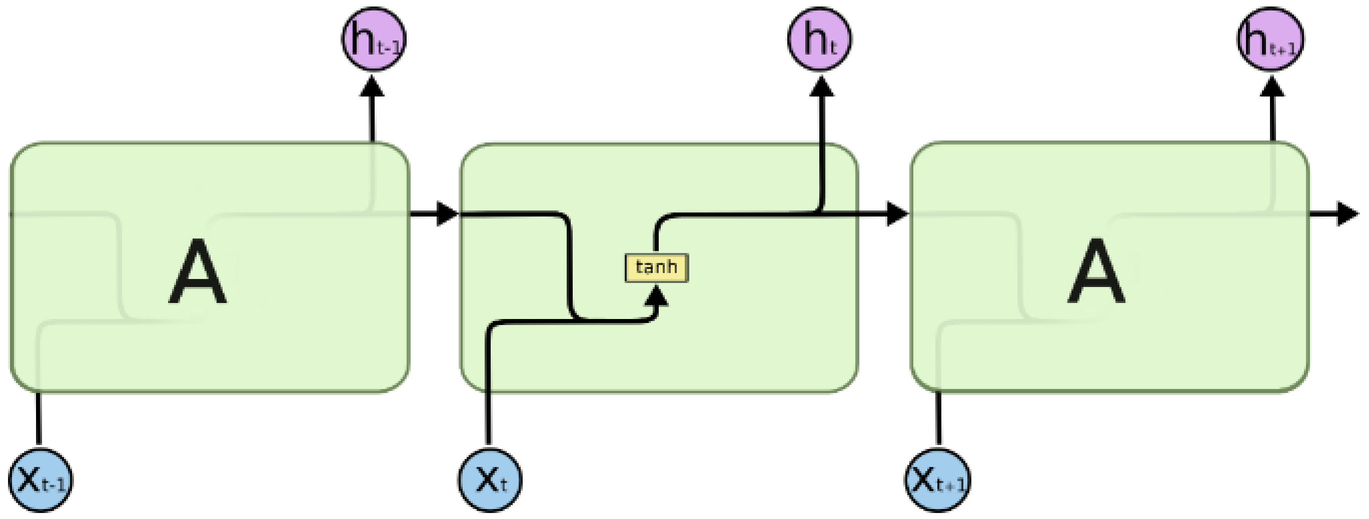
Material from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Reccurent Neural Network



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

Reccurent Neural Network



- RNNs work well with recent information

Example:

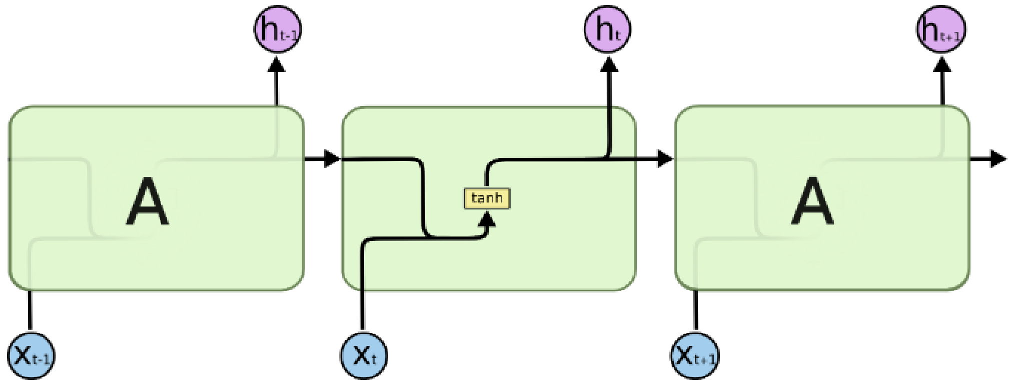
- Predict the last word: "The clouds are in the **sky**"

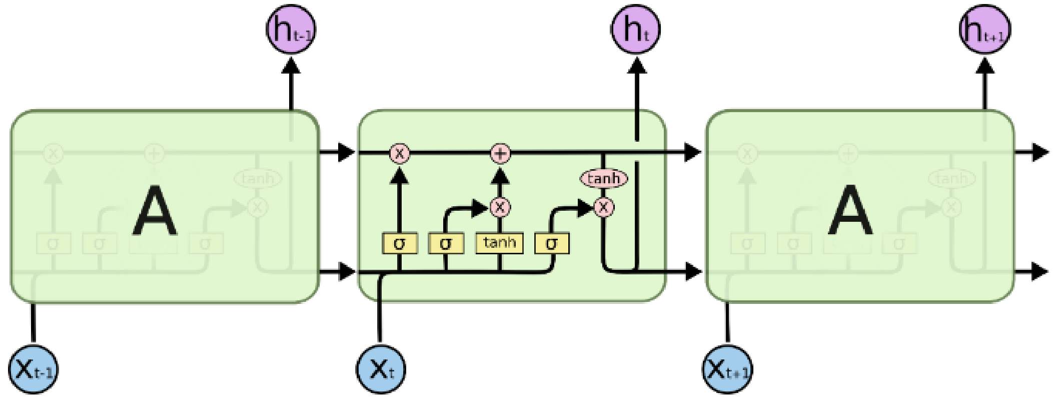
But consider predicting the last word in

- "I grew up in France… I speak fluent **French.**"

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

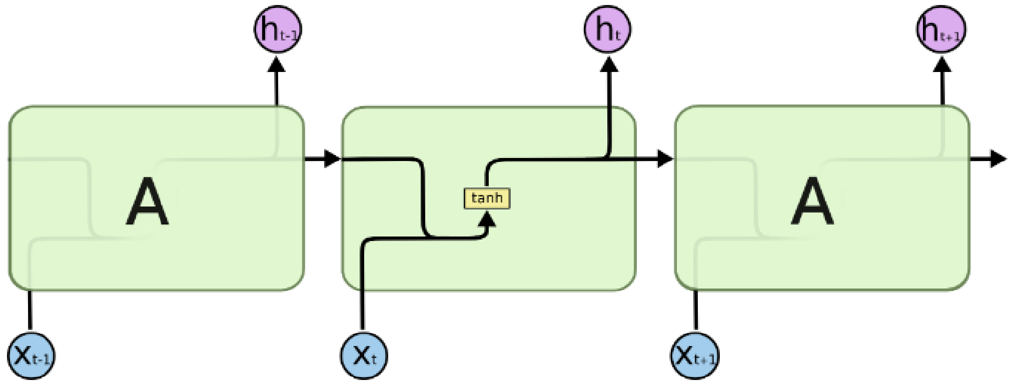
What is the difference between RNNs and LSTMs?

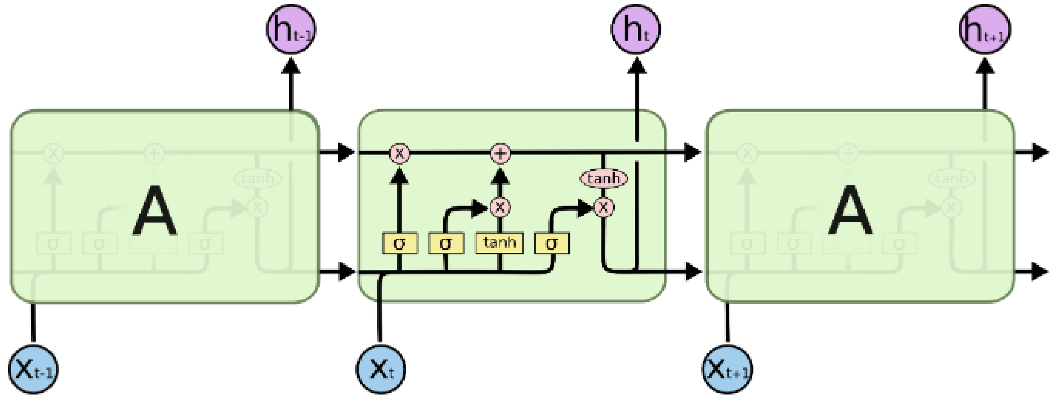




Material from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

What is the difference between RNNs and LSTMs?





- LSTMs are a special version of RNNs. Most RNN success stories involve LSTMs
- It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.
- Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.

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

LSTMs are a special version of RNNs
Most RNN success stories involve LSTMs

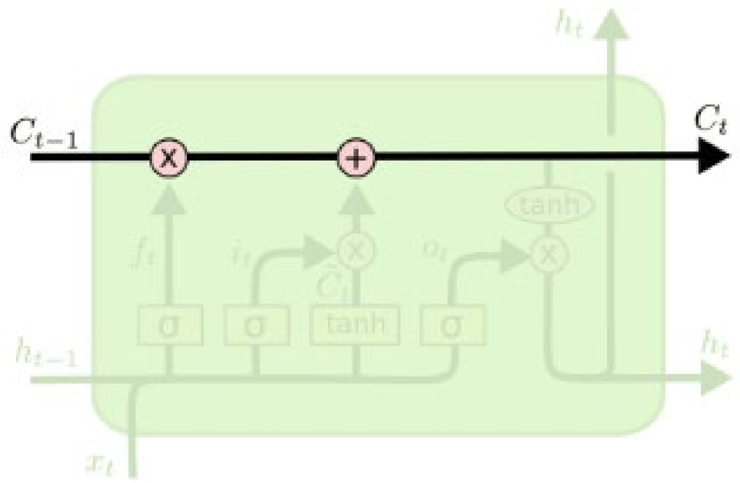
- In theory, RNNs are absolutely capable of handling such “long-term dependencies.”
  - A human could carefully pick parameters for them to solve toy problems of this form.
  - Sadly, in practice, RNNs don’t seem to be able to learn them.
  - The problem was explored in depth by Hochreiter (1991) [German] and Bengio, et al. (1994), who found some pretty fundamental reasons why it might be difficult.

- Thankfully, LSTMs don’t have this problem!

- Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

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

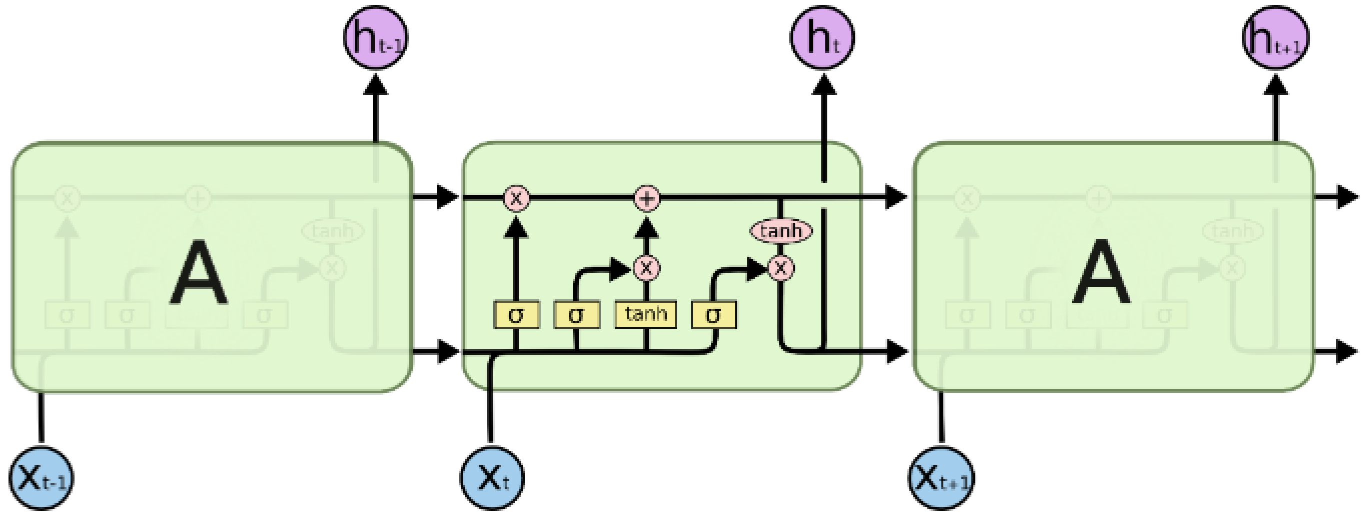
The core idea behind LSTM: the cell state



- The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged

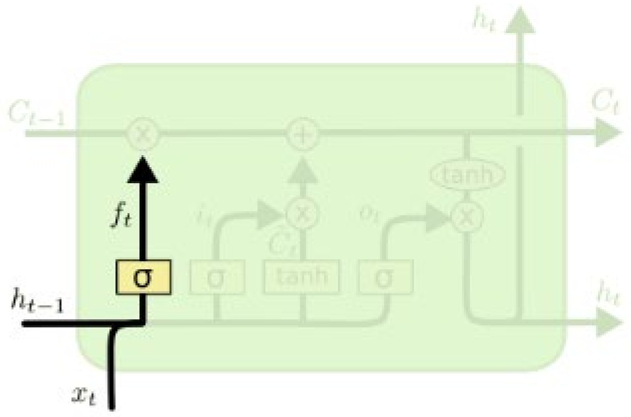
Material from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long short-term memory neural networks



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

The “forget gate”

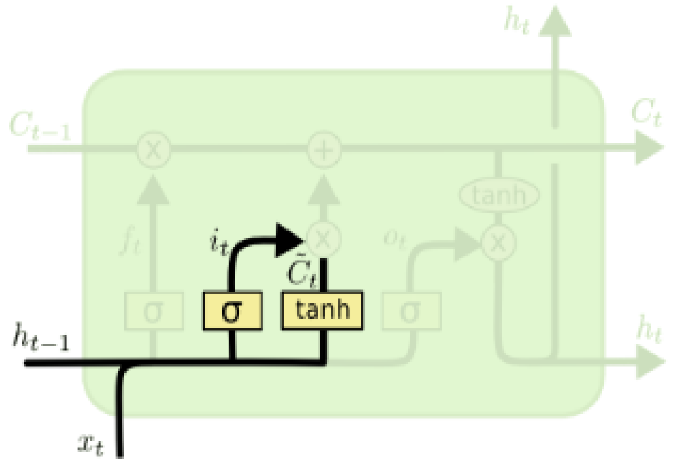


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

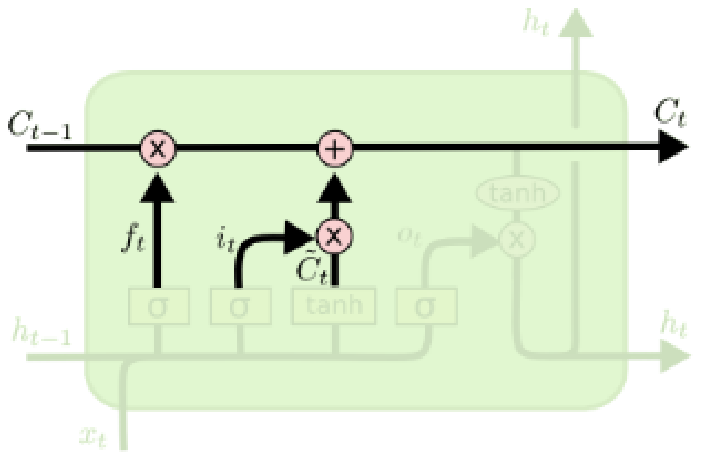
- The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.

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

The “update gate”



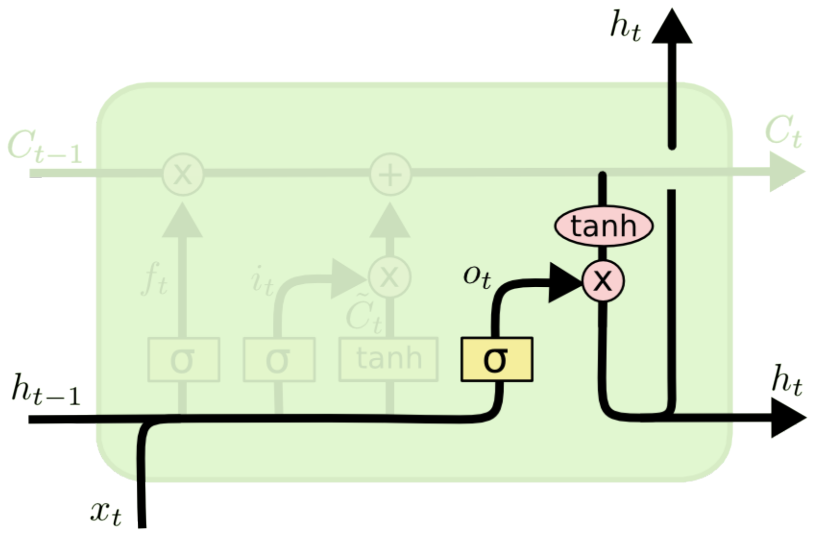
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$



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

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

The “output gate”

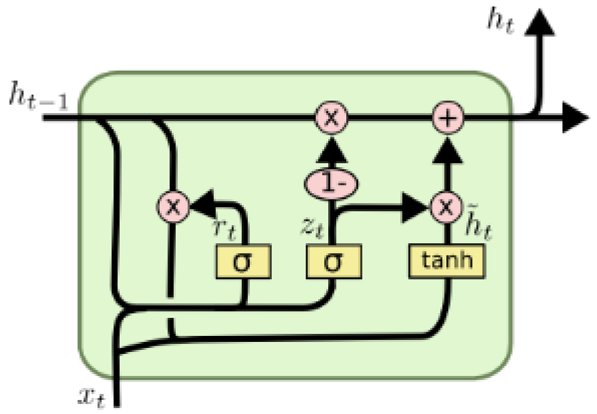


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- The output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we’re going to output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

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

The LSTM



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

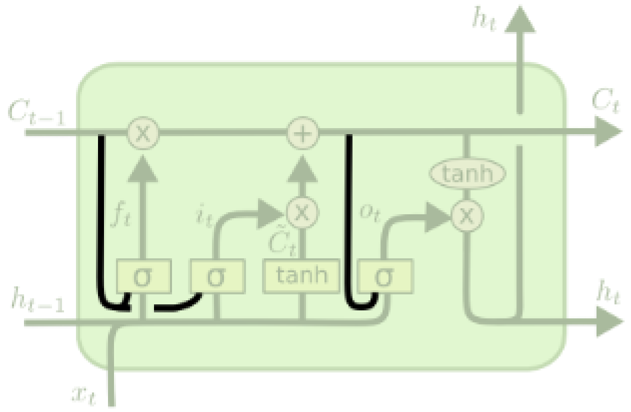
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

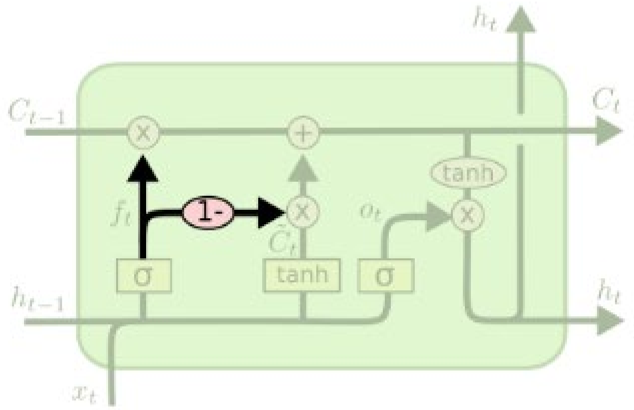
Material from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Variants



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \;+\; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \;+\; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \;+\; b_o\right)$$

- Peephole connections

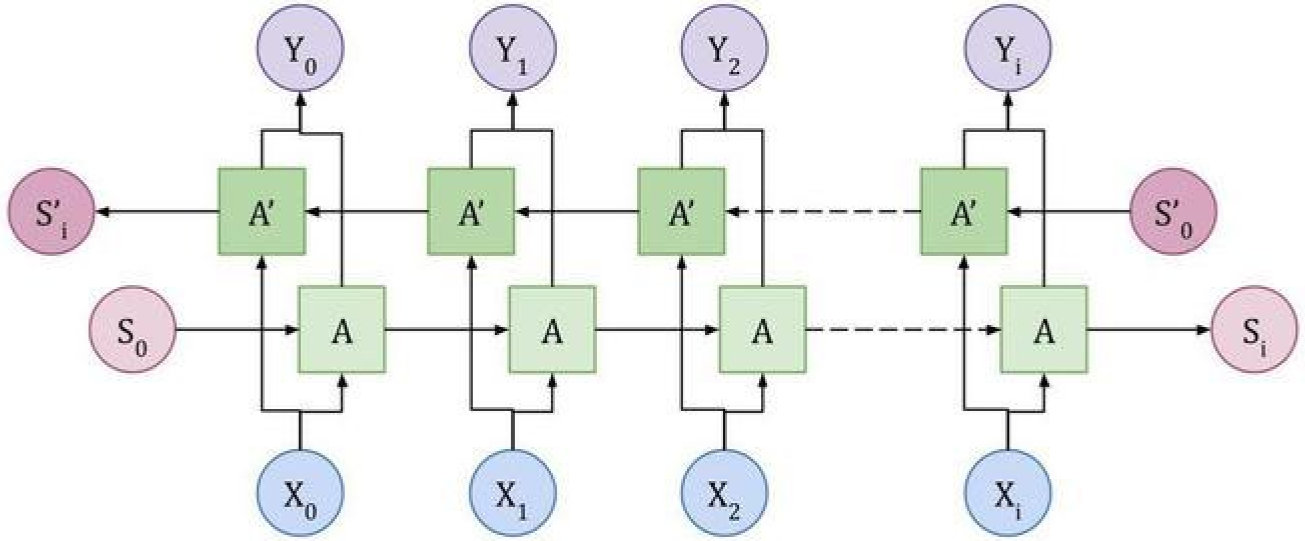


$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

- Coupled input and forget

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

Bidirectional LSTMs

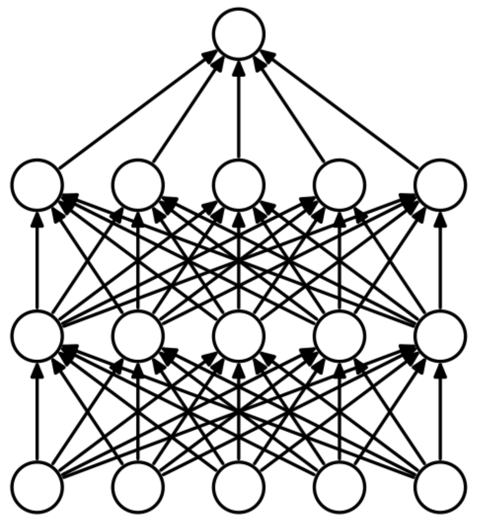


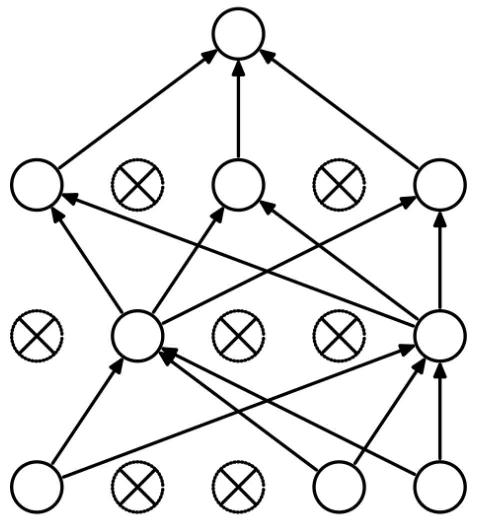
- Bidirectional LSTM or BiLSTM is a term used for a sequence model which contains two LSTM layers, one for processing input in the forward direction and the other for processing in the backward direction. It is usually used in NLP-related tasks. The intuition behind this approach is that by processing data in both directions, the model is able to better understand the relationship between sequences (e.g. knowing the following and preceding words in a sentence).

- This architecture can be interpreted as having two separate LSTM networks, one gets the sequence of tokens as it is while the other gets in the reverse order. Both of these LSTM network returns a probability vector as output and the final output is the combination of both of these probabilities.

Material from: https://www.geeksforgeeks.org/bidirectional-lstm-in-nlp/

Dropout


(a) Standard Neural Net


(b) After applying dropout.

- Dropout prevents overfitting
- Dropout refers to dropping the input of a neuron → implies zero input.
- Whether a neuron drops its input or not is decided by the dropout rate P(drop).
  - If P(drop) = 0.8, each input randomly chooses a number from 0 to 1. If the chosen number is lesser than 0.8, that output will be dropped.
- A dropout on the input means that for a given probability, the data on the input connection to each LSTM block will be excluded from node activation and weight updates.

Source: https://tigerprints.clemson.edu/cgi/viewcontent.cgi?article=3918&context=all_theses

Revisit after this lecture: Hands-on of March 20

- What are the main takeaways now?

- How does the NN structure look like?
- What is the activation function used?
- Scaling?
- Dropout?