LSTM

COMPUTATIONAL LINGUITICS

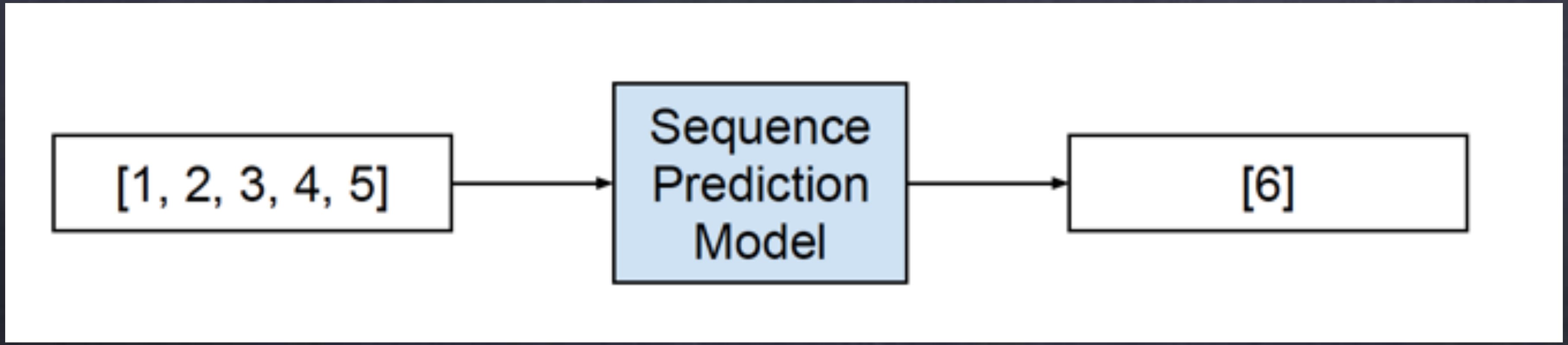
HYopil Shin
hpshin@snu.ac.kr

What are LSTMs

- Sequence Prediction Problems
  - different to other types of supervised learning problems
  - the sequence imposes an order on the observations that must be preserved when training models and making predictions
- Four different types of sequence predictions
  - Sequence Prediction
  - Sequence Classification
  - Sequence Generation
  - Sequence-to-Sequence Prediction

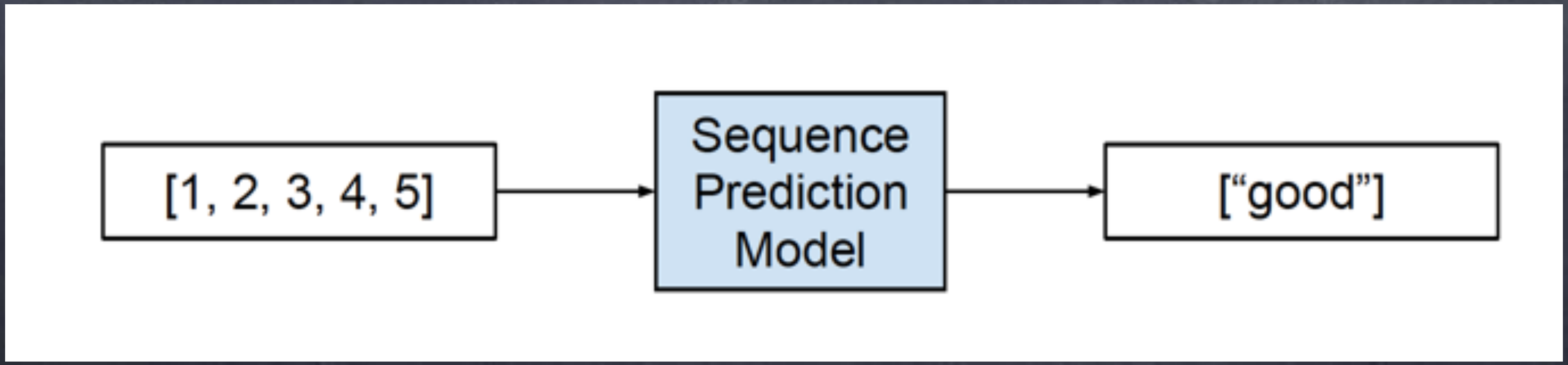
Sequence

- The Sequence imposes an explicit order on the observations!
- The order is important!
- Sequence Prediction
  - Weather Forecasting: given a sequence of observations about the weather over time, predict the expected weather tomorrow
  - Stock Market Prediction: given a sequence of movements of a security over time, predict the next movement of the security
  - Product Recommendation: given a sequence of past purchases for a customer, predict the next purchase for a customer



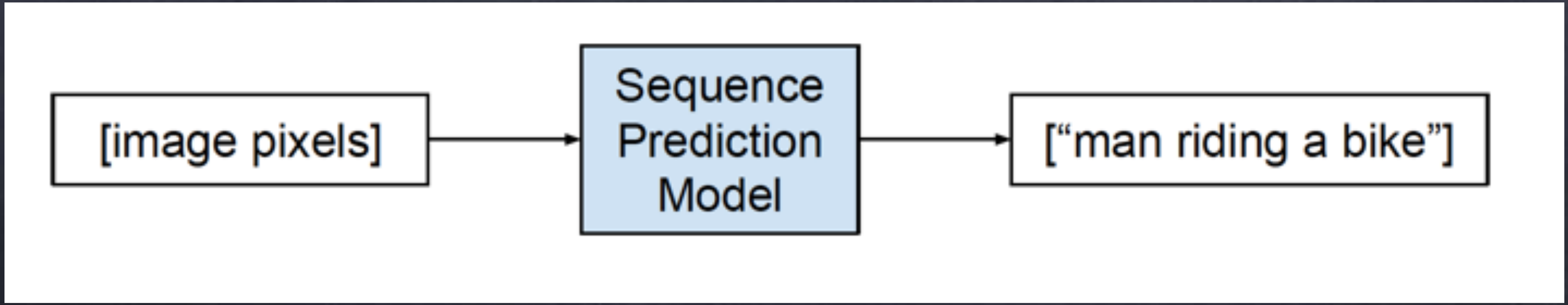
Sequence Classification

- sequence classification involves predicting a class label for a given input sequence
  - DNA Sequence Classification: given a DNA sequence of A, C, G and T values, predict whether the sequence is for a coding or non-coding region
  - Anomaly Detection: given a sequence of observations, predict whether the sequence is anomalous or not
  - Sentiment Analysis: given a sequence of text such as a review or a tweet, predict whether the sentiment of the text is positive or negative



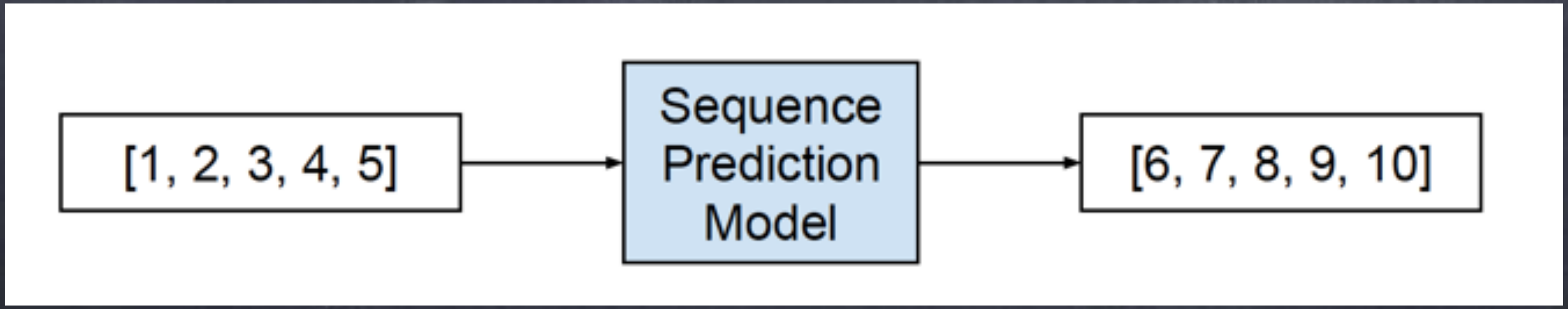
Sequence Generation

- Sequence generation involves generating a new output sequence that has the same general characteristics as other sequences in the corpus
  - Text Generation: given a corpus of text, such as works of Shakespeare, generate new sentences or paragraphs of text that read they could have been drawn from the corpus
  - Handwriting Prediction: given a corpus of handwriting examples, generate handwriting for new phrases that has the properties of handwriting in the corpus
  - Music Generation: given a corpus of examples of music, generate new musical pieces that have the properties of the corpus
  - Image Caption Generation: given an image as input, generate a sequence of words that describe image



Sequence-to-Sequence Prediction

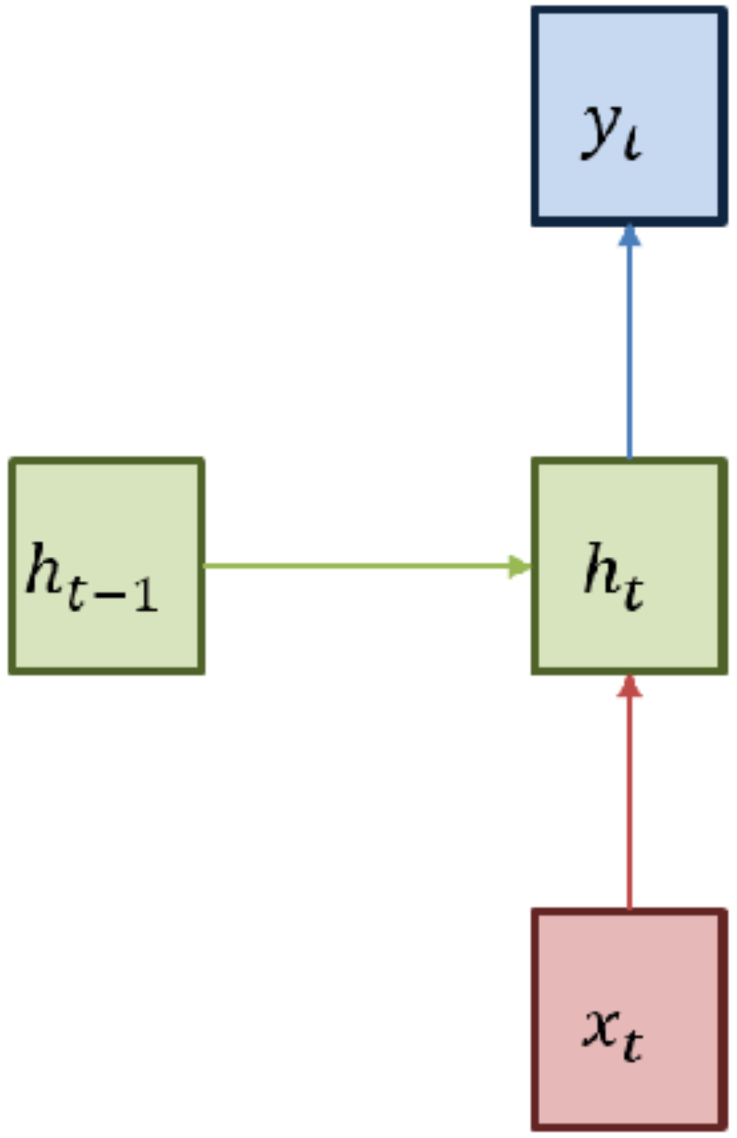
- Sequence-to-sequence prediction involves predicting an output sequence given an input sequence
  - Multi-step Time Series Forecasting: given a time series of observations, predict a sequence of observations for a range of future time steps
  - Text Summarization: given a document of text, predict a shorter sequence of text that describes the sailing parts of the source documents
  - Program Execution: given the textual description program or mathematical equation predict the sequence of characters that describes the correct output



Limitations of Multilayer Perceptrons

- Stateless- MLPs learn a fixed function approximation. Any outputs that are conditional on the context of the input sequence must be generalized and frozen into the network weights
- Unaware of Temporal Structure- Time steps are modeled as input features, meaning that network has no explicit handling or understanding of the temporal structure or order between observations
- Messy Scaling- For problems that require modeling multiple parallel input sequences the number of input features increases as a factor of the size sliding window without any explicit separation of time steps of series
- Fixed Size Inputs- The size of the sliding window is fixed and must be imposed on all inputs to the network
- Fixed Size Outputs- The size of the output is also fixed and any outputs that do not conform must be forced

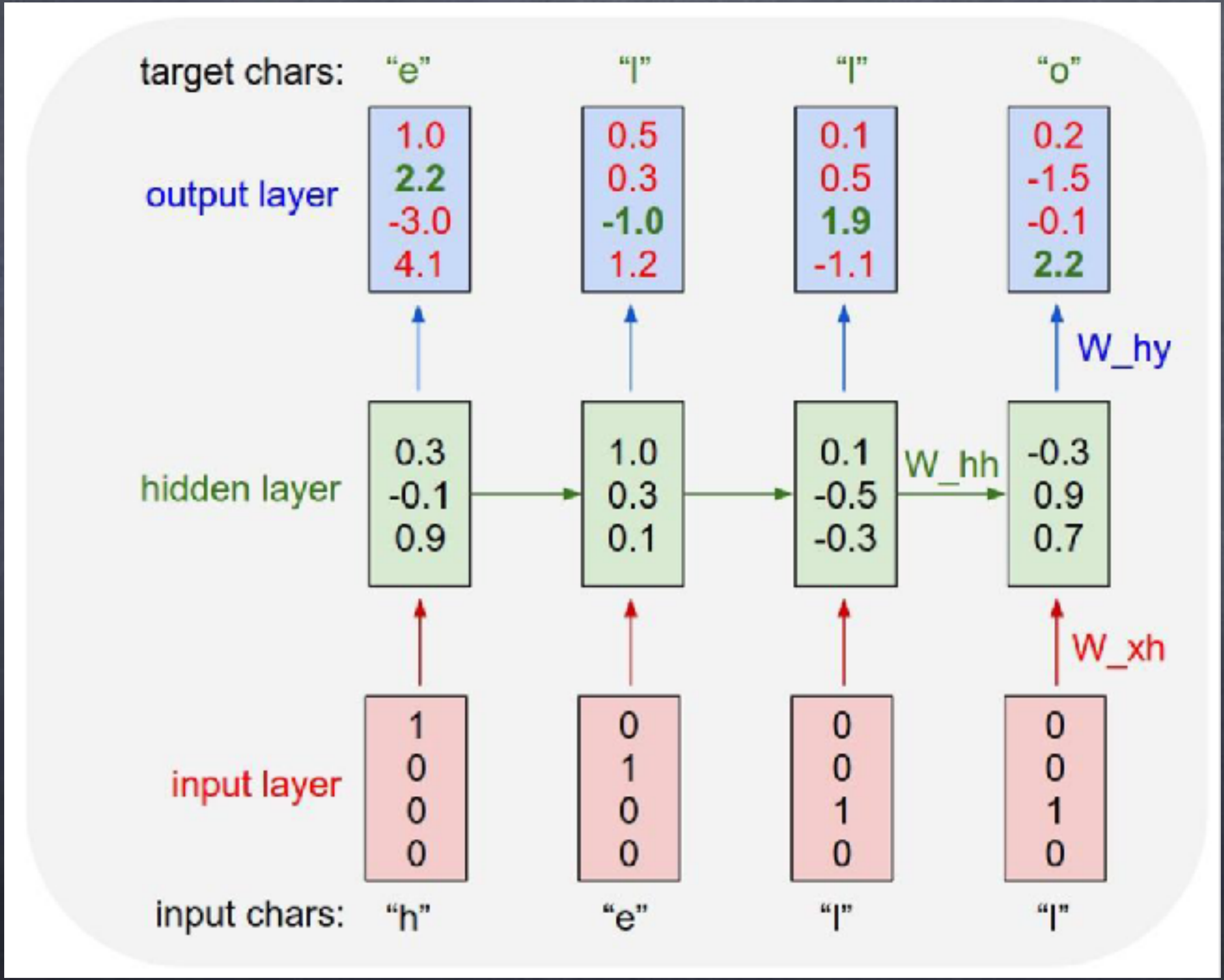
RNN: Revisited



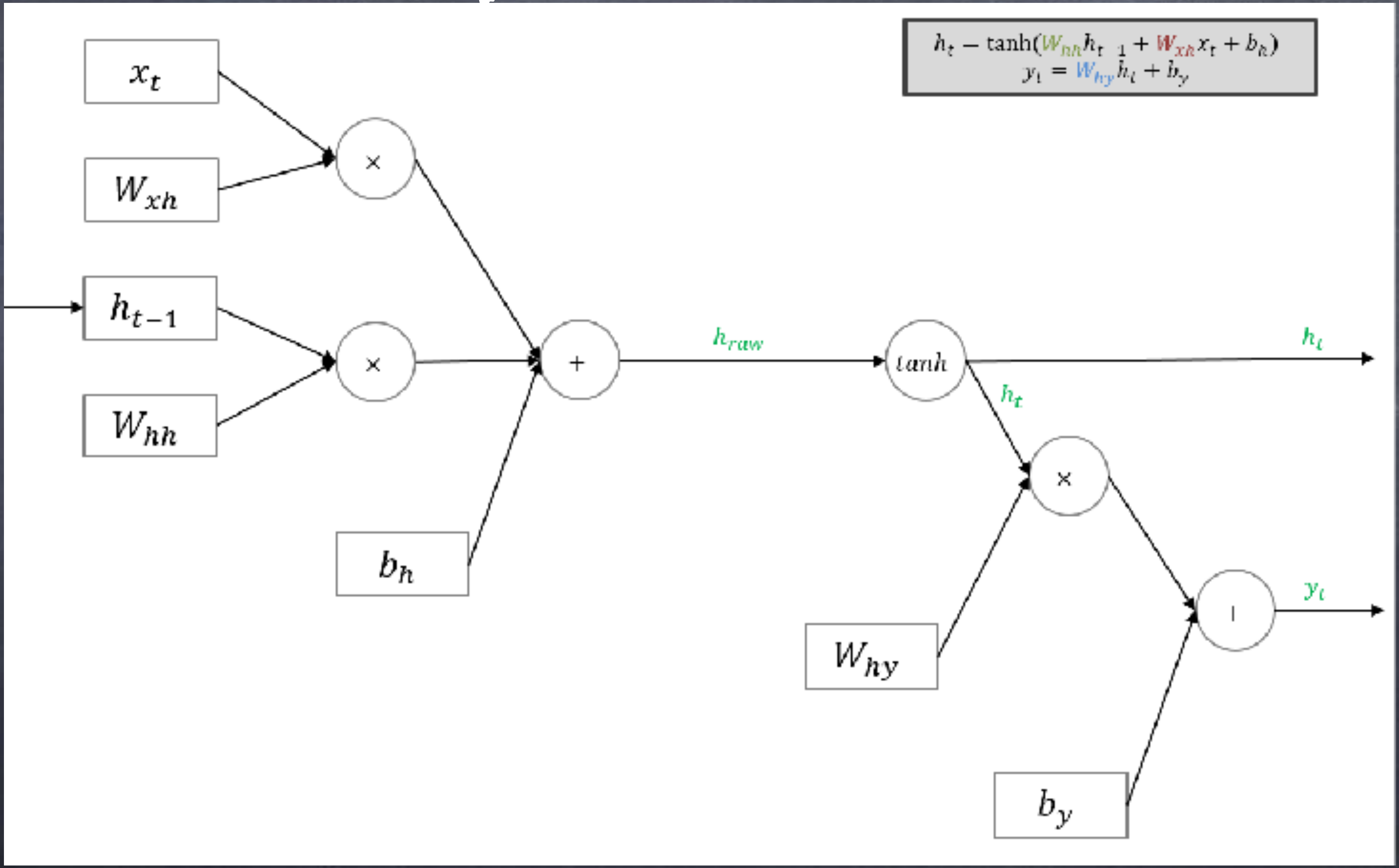
$$y_t = W_{hy} h_t + b_y$$

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + b_h)$$

RNN : Revisited

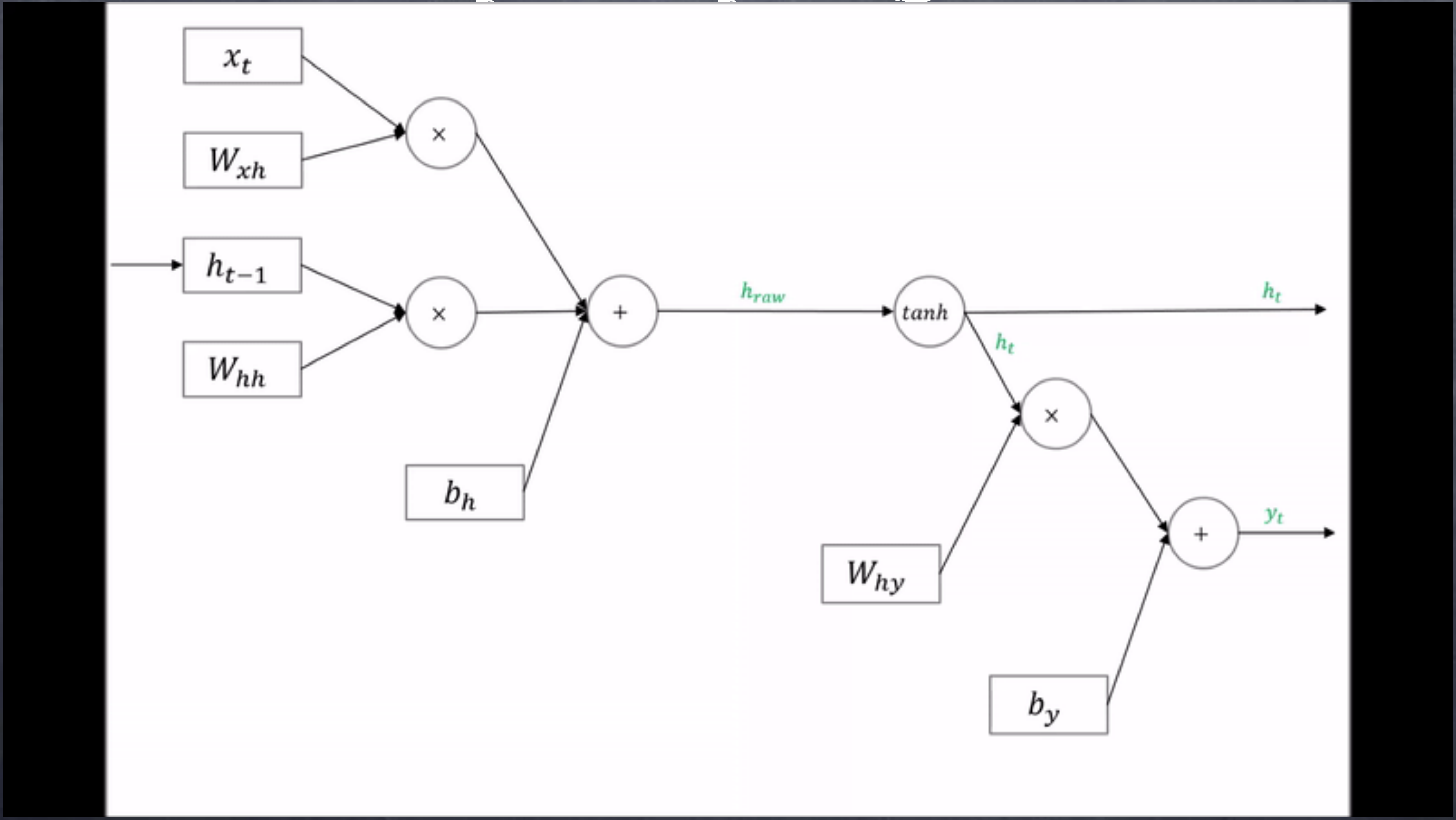


RNN - forward Propagation



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$
$$y_t = W_{hy}h_t + b_y$$

RNN-Backpropagation



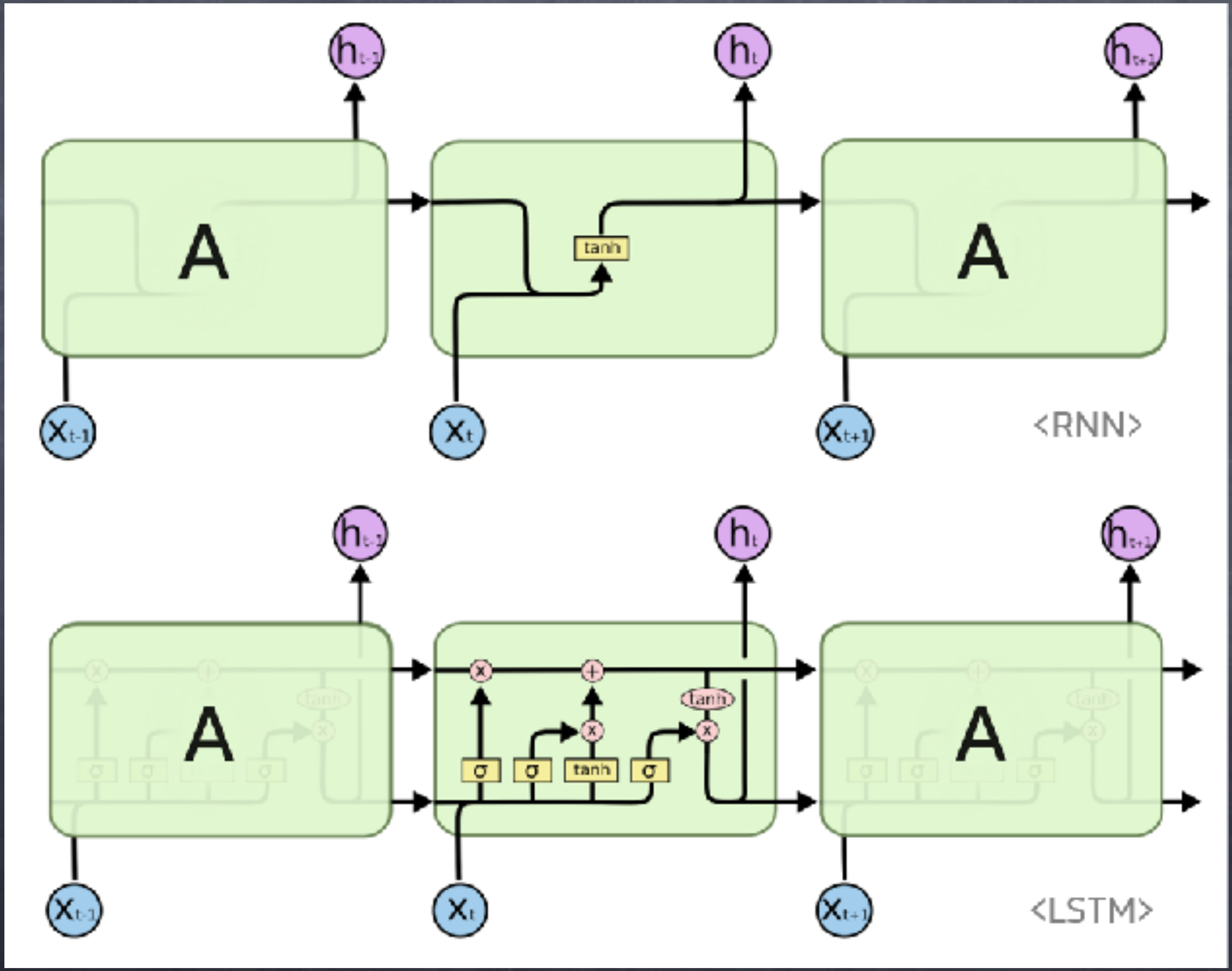
The Long Short-Term Memory Network

- Given a standard feedforward MLP network, an RNN can be thought of as the addition of loops to the architecture
- The recurrent connections add state or memory to the network and allow it to learn and harness the ordered nature of observations within input sequences
- Like RNNs, the LSTMs have recurrent connections so that the state from previous activations of the neuron from the previous time step is used as context for formulating an output
- Unlike other RNNs, the LSTM has a unique formulation that allows it to avoid the problems that prevent the training and scaling of the other RNNs
- The key historical challenge faced with RNNs is how to train them effectively.
- Experiments show how difficult this was where the weight update procedure resulted in weight changes that quickly became so small as to have no effect (vanishing gradient) or so large as to result in very large changes or even overflow (exploding gradients)
- LSTMs overcome this challenge by design
- The computational unit of the LSTM network is called memory cell, memory block, or just cell for short

The Long Short-Term Memory Network

- LSTM cells are comprised of weights and gates
  - LSTM Weights : A memory cell has weight parameters for input, output, as well as internal state
    - Input Weights: used to weight input for the current time step
    - Output Weights: used to weight the output from the last time step
    - Internal State: internal state used in the calculation of the output for this time step
  - LSTM Gates: These too are weighted functions that further govern the information flow in the cell
    - Forget Gate: decides what information to discard from the cell
    - Input Gate: decides which values from the input to update the memory state
    - Output Gate: decides what to output based on input and the memory of the cell
  - The forget and input gate are used in the updating of the internal state. The output gate is a final limiter on what the cell actually outputs

LSTM



h_{t-1}
h_t
h_{t+1}
A
tanh
A
x_{t-1}
x_t
x_{t+1}
<RNN>
h_{t-1}
h_t
h_{t+1}
A
σ
σ
tanh
σ
tanh
A
x_{t-1}
x_t
x_{t+1}
<LSTM>

The Long Short-Term Memory Network

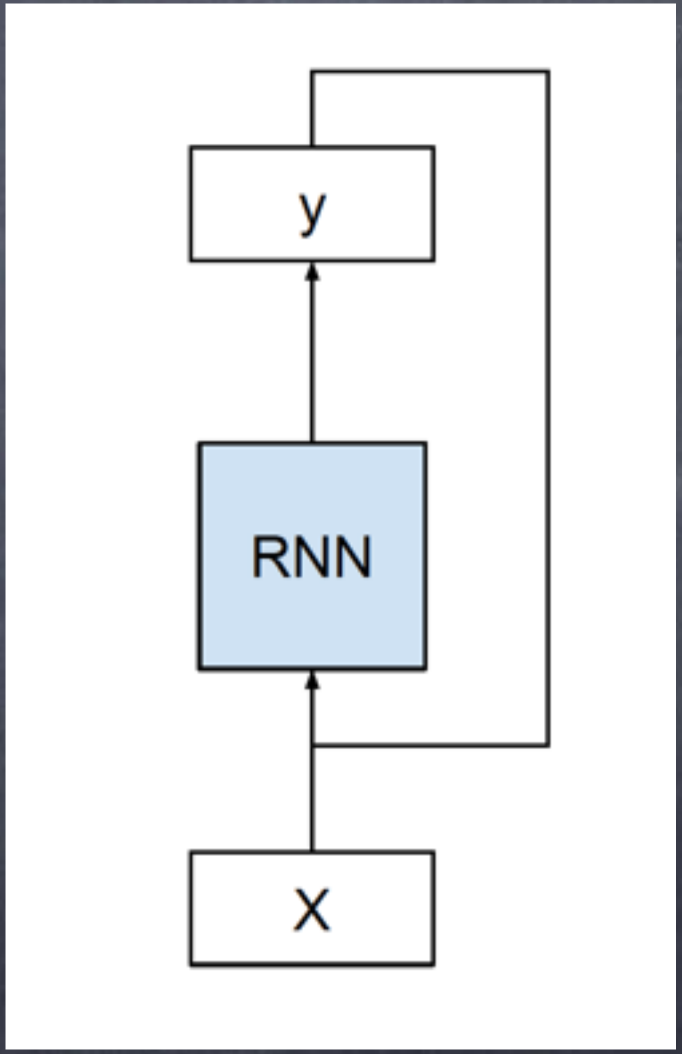
- Three Key benefits of LSTMs
    - Overcomes the technical problems of training an RNN, namely vanishing and exploding gradients
    - Possesses memory to overcome the issues of long-term temporal dependency with input sequences
    - Process input sequences and output sequences time step by time step, allowing variable length inputs and outputs
- Limitations of LSTMs
    - Memory: how memory can be abused.
    - remembering a single observation over a very long number of input time steps is a poor use of LSTMS
    - requiring an LSTM model to remember multiple observations will fail

How to Train LSTMs

- Backpropagation Training Algorithm
    - The mathematical method used to calculate derivatives and an application of the derivative chain rule
    - The training algorithm for updating network weights to minimize error
- The goal of the back propagation training algorithms is to modify the weights of a neural network in order to minimize the error of the network outputs compared to some expected output in response to corresponding inputs
- General Backpropagation algorithm
    - Present a training input pattern and propagate it through the network to get an output
    - Compare the predicted outputs to the expected outputs and calculate the error
    - Calculate the derivatives of the error with respect to the network weights
    - Adjust the weights to minimize the error
    - Repeat

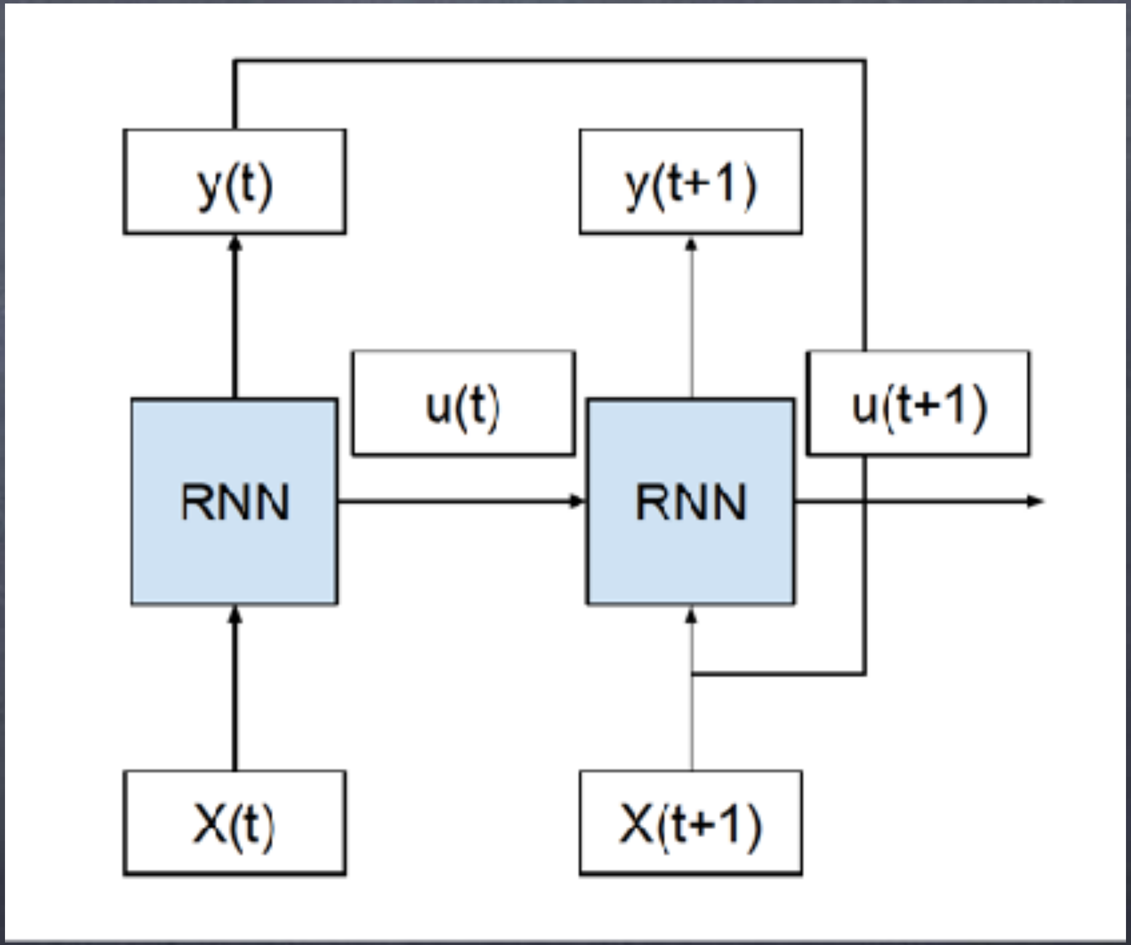
Unrolling Recurrent Neural Networks

- A simple conception of recurrent neural network is a type of neural network that takes inputs from previous time steps
- RNNs are fit and make predictions over many time steps.
- As the number of time steps increases, the simple diagram with a recurrent connection begins to lose all meaning



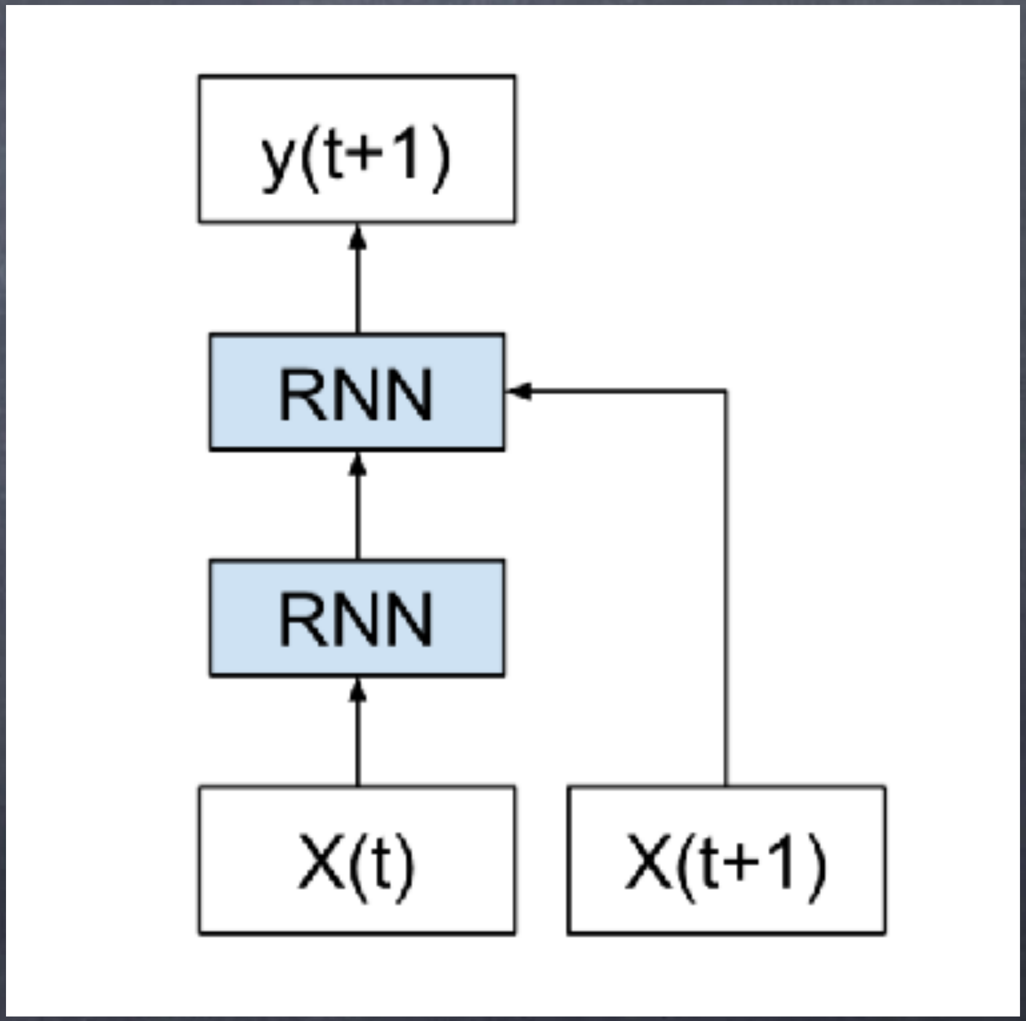
Unfolding the Forward Pass

- For multiple time steps of input(x(t), x(t+1) ...), internal state(u(t), u(t+1)...), and outputs(y(t), y(t+1)...), we can unfold the network schematic into a graph without any cycles
- The cycle is removed and the output (y(t)) and infernal state (u(t)) from the previous time step are passed onto the network as inputs for processing the next time step
- The network does not change between the unfolded time steps. Specifically, the same weights are used for each time step and it is only the outputs and the internal states that differ
- In this way, the whole network are copied for each time step in the input sentence

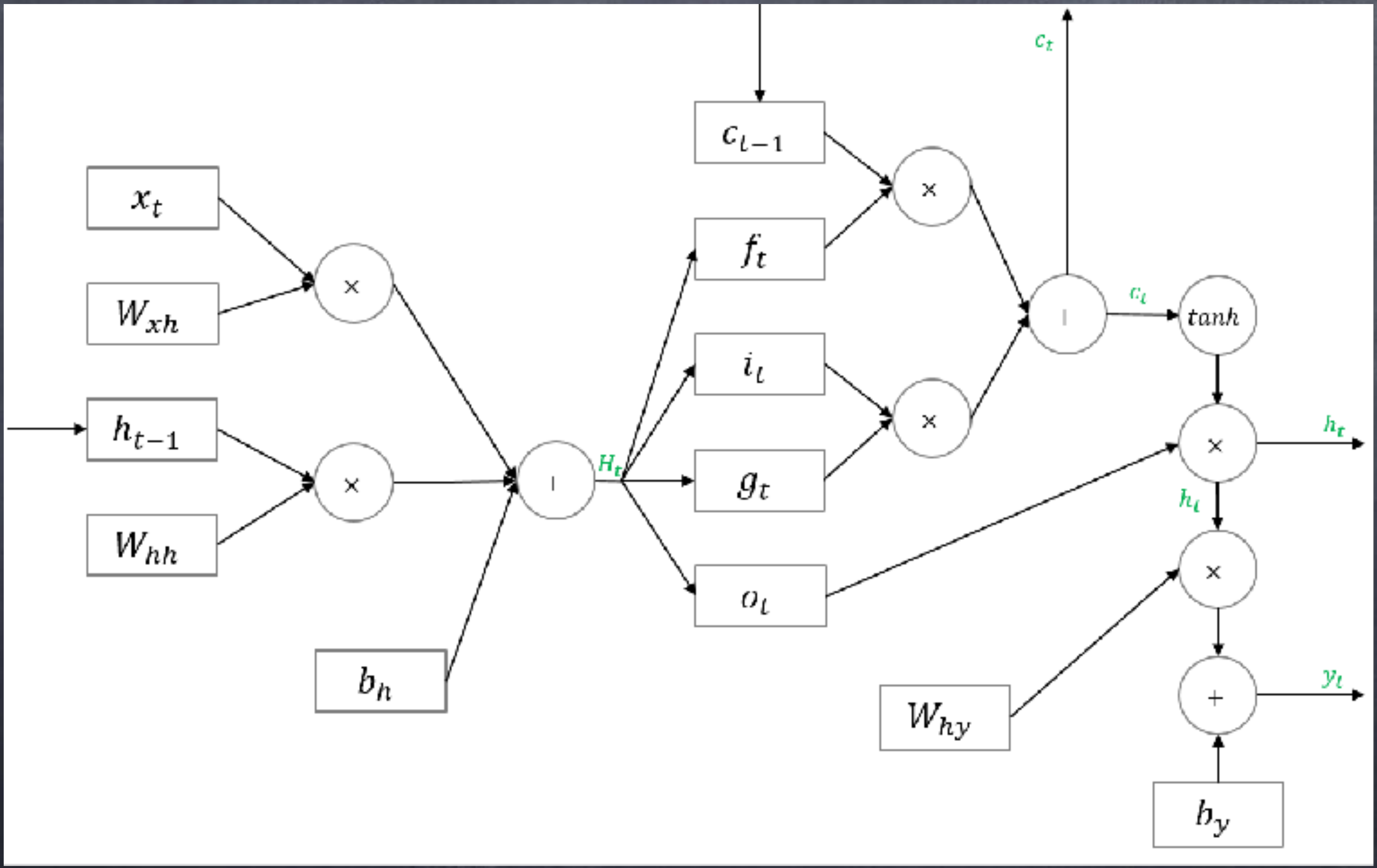


Unfolding the forward pass

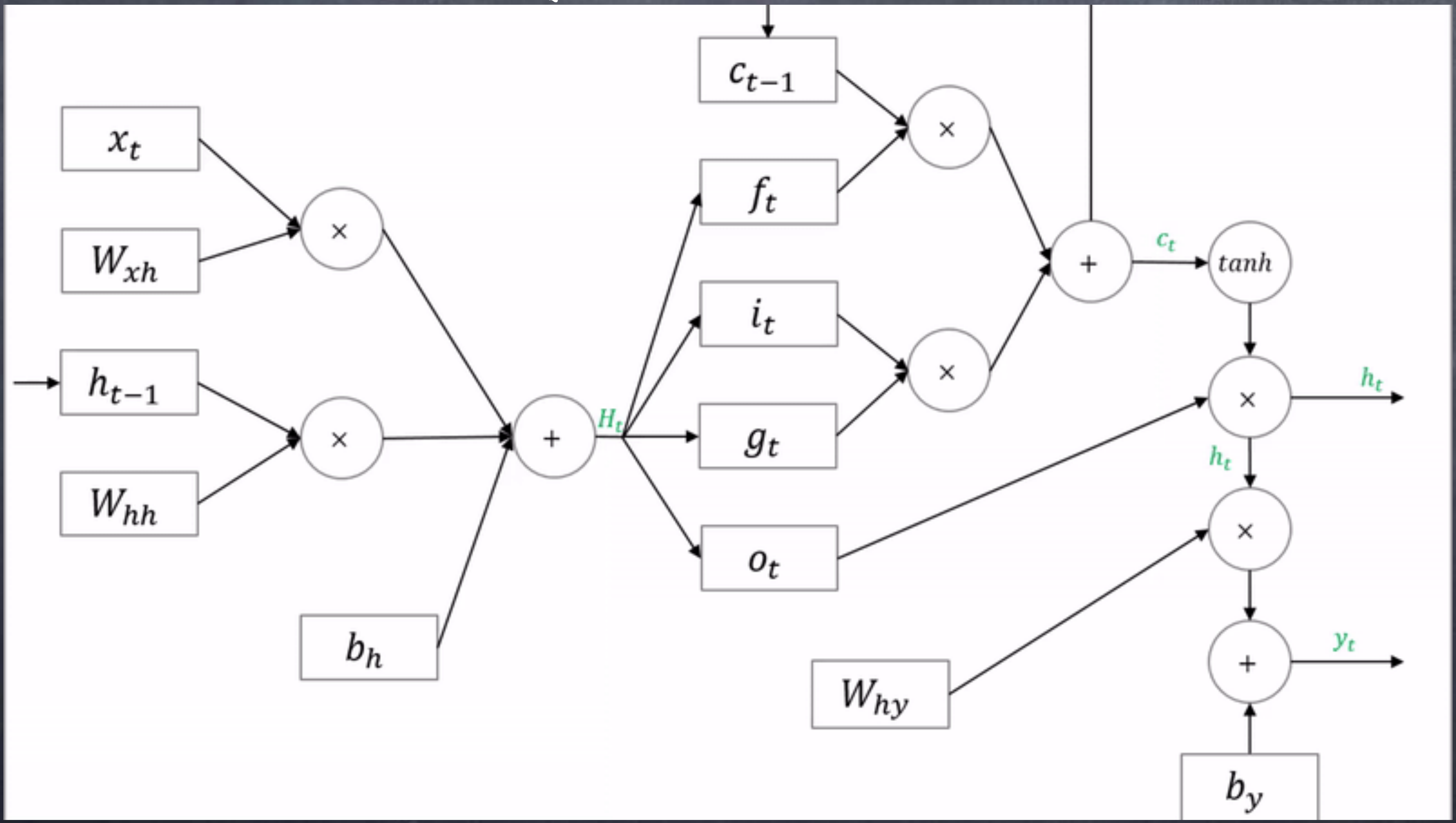
- Each copy of the network may be thought of as an additional layer of the same forward neural network
- The deeper layers take as input the output of the prior layer as we'll as a new input time step
- The layers are in fact all copies of the same set of weights and the internal state is updated from layer to layer, which may be a stretch of this oft-used analogy
- RNNs, once unfolded in time, can be seen as very deep feedforward networks in which all the layers share the same weights



LSTM-Forward Propagation



LSTM-Back Propagation



Unfolding the Backward Pass

- The idea of network unfolding plays a bigger part in the way recurrent neural networks are implemented for the backward pass
- The backpropation of error for a given time step depends on the activation of the network at the prior time step
- The backward pass requires the conceptualization of unfolding the network
- Error is propagated back to the first input time step of the sequence so that the error gradient can be calculated and the weights fo the network can be updated
- Additional concerns of unfolding the recurrent network graphs:
    - Each time step requires a new copy of the network which in turn takes more memory, especially for large networks with thousands of millions of weights
    - The memory requirements of training large recurrent networks can quickly balloon as the number of time steps climbs into the hundreds

Backpropagation Through Time(BPTT)

- BPTT is the application of the Backpropagation training algorithm to RNN
- BPTT works by unrolling all input time steps. Each time step has one input time step, one copy of the network, and one output
- Errors are then calculated and accumulated for each time step
- summarize the algorithm as follows:
    - Present a sequence of time steps of input and output to the network
    - Unroll the network then calculate and accumulate errors across each time step
    - Roll-up the network and update weights
    - Repeat
- BPTT can be computationally expensive as the number of time steps increases. If input sentences are comprised of thousands of time steps, then this will be the number of derivatives required for a single weight update
- This can cause weights to vanish or explode and make slow learning and model skill noisy

Truncated Backpropagation Through Time(TBPTT)

- TBPTT is a modified version of the BPTT training algorithm for recurrent neural network where the sequence is processed over the time step at a time and periodically an update is performed back for a fixed number of time steps
- The algorithms can be summarized as follows:
  - Present a sequence of k1 time steps of input and output pairs to the network
  - Unroll the network, then calculate and accumulate errors across k2 time steps
  - Roll-up the network and update weights
  - Repeat
- TBPTT algorithm requires the consideration of two parameters
  - k1: The number of forward-pass time steps between updates. Generally this influences how slow or fast training will be, given how often weight updates are performed
  - k2: The number of time steps to which to apply to BPTT. Generally, it should be large enough to capture the temporal structure in the problem for the network to learn

Configuration for Truncated BPTT

- Following Sutskever's k1 and k2 parameters, some standard or common notations can be adapted.(here n refers to the total number of time steps in the input sentence)
  - TBPTT(n, n): Updates are performed at the end of the sequence across all time steps in the sequence(e.g. classical BPTT)
  - TBPTT(1, n): time steps are processed one at time followed by an update that covers all time steps seen so far (e.g. classical TBPTT by Williams and Peng)
  - TBPTT(k1, 1): The network likely does not have enough temporal context to learn, relying heavily on internal state and inputs
  - TBPTT(k1, k2), where k1<k2<n: Multiple updates are performed per sequence which can accumulate training
  - TBPTT(k1, k2), where k1=k2: A common configuration where a fixed number of time steps is used for both forward and backward-pass time steps (e.g. 10s to 100s)
- Canonical TBPTT reported in papers may be considered TBPTT(k1, k2), where k1=k2=k and k<=n. K is a single parameter and often claimed that the sequence length of input time steps should be limited to 200-400