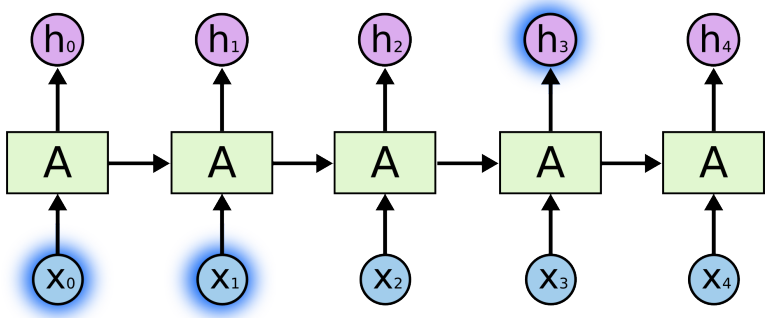
Understanding LSTM Networks

Outline

- Why Long Short Term Memory Networks (LSTM)?
- Adding “gates” to an RNN
  - prevent the vanishing gradient problem.
- Train LSTMs/GRUs to retain state information longer and handle long-distance dependencies.

Vanishing Gradient Problem

The problem of Long-Term Dependencies:



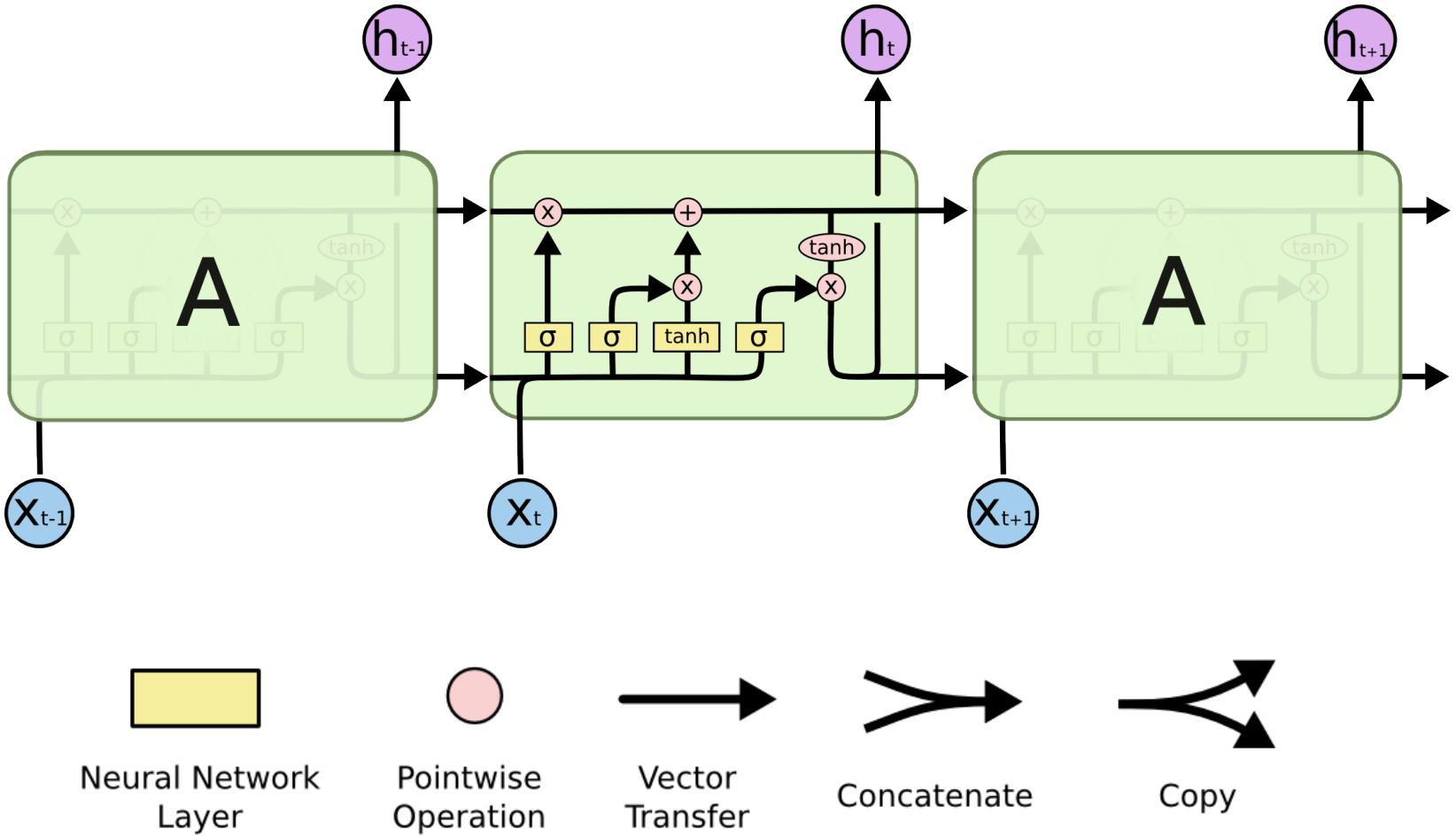
- Backpropagated errors multiply at each layer, resulting in exponential decay (if derivative is small) or growth (if derivative is large)

- It makes it very difficult train deep networks, or simple recurrent networks over many time steps

Long Short Term Memory

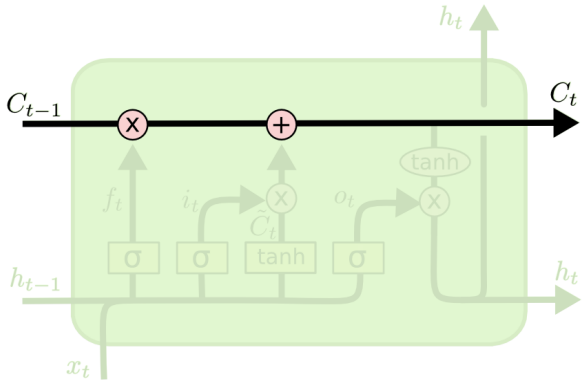
- LSTM networks, add additional gating units in each memory cell.
  - Forget gate
  - Input gate
  - Output gate
- Prevents vanishing gradient problem and allows network to retain state information over longer periods of time.

LSTM Network Architecture



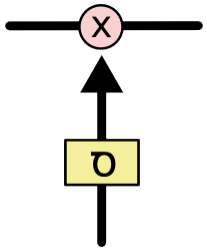
Core Idea Behind LSTM Network

- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.



- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

- Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

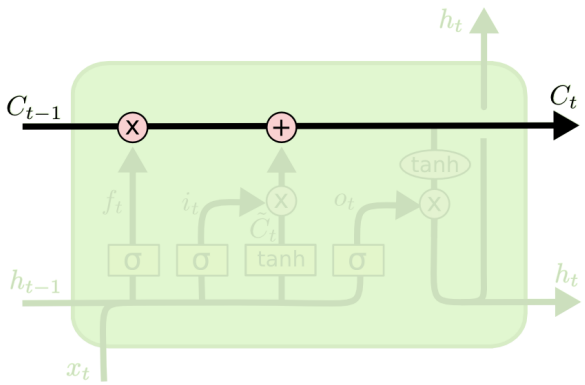


LSTM Concept

Long Term Memory

Short Term Memory

Output



Input

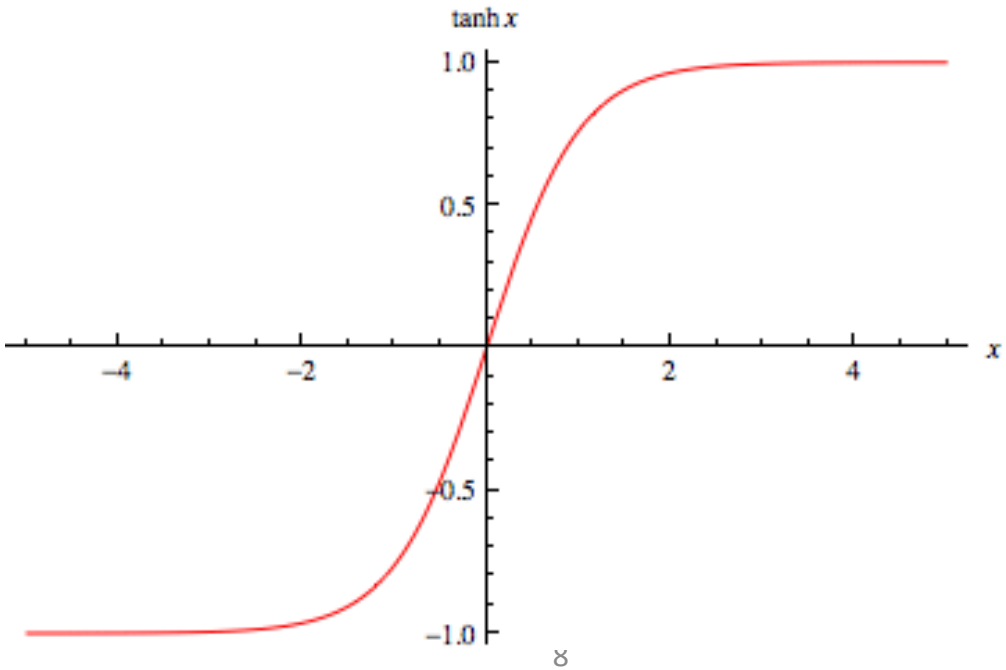
Long Term Memory

Short Term Memory

Hyperbolic Tangent

- Tanh can be used as an alternative nonlinear function to the sigmoid logistic (0-1) output function.

- Used to produce thresholded output between –1 and 1.



Gates

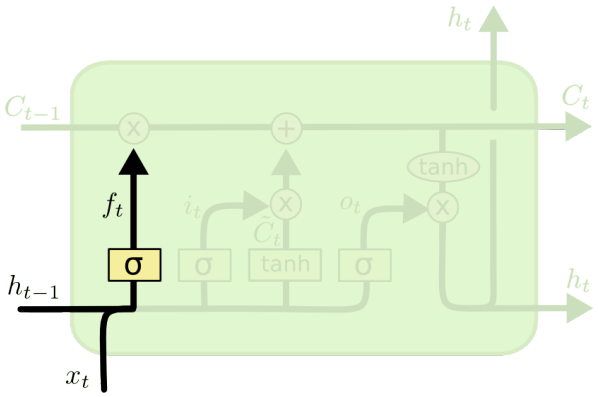
- Forget gate will remove existing information of a prior subject when a new one is encountered.

- Input gate adds in the information

Forget Gate

- The first step in our LSTM is to decide **what information we're going to throw away from the cell state**. This decision is made by a sigmoid layer called the "**forget gate layer**."

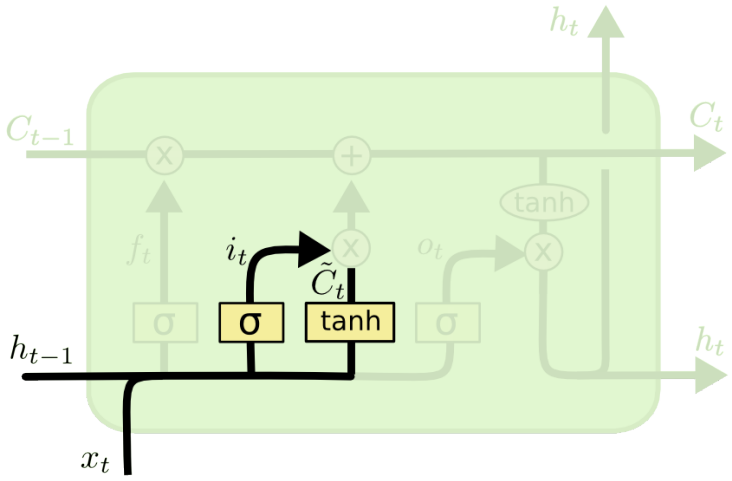
- It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."



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

Input Gate

- The next step is to decide what new information we're going to store in the cell state. This has two parts.
  - First, a sigmoid layer called the "input gate layer" decides **which values we'll update**.
  - Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.
  - In the next step, we'll combine these two to create an update to the state.

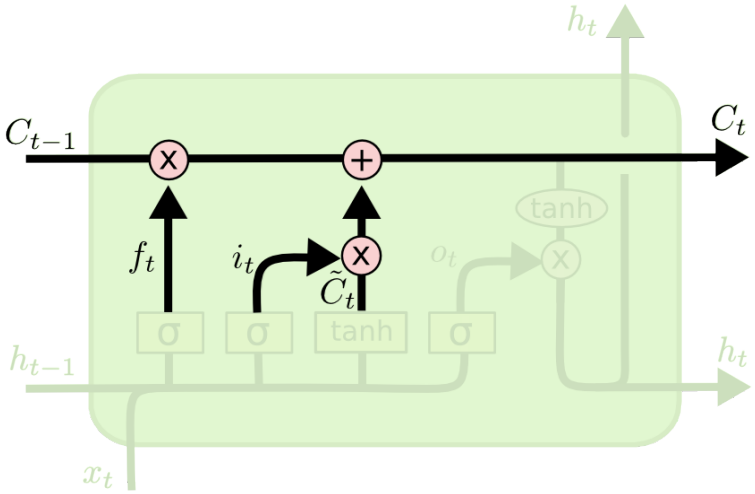


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Update the old cell state (remember)

- It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.
- We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

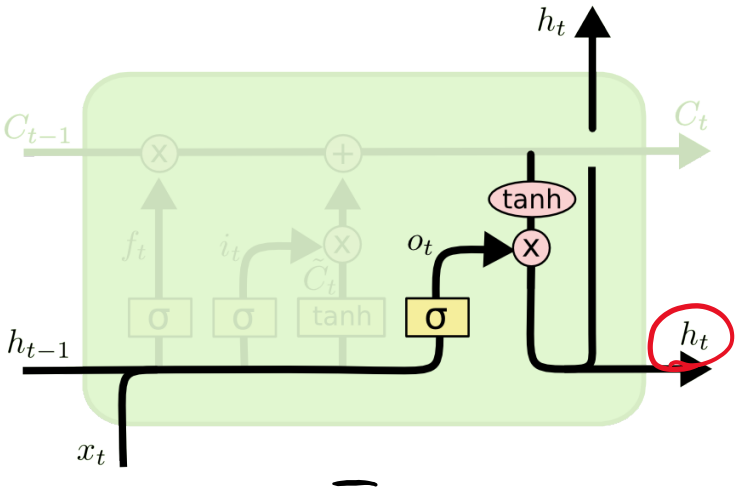


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output

Finally, we need to decide what we're going to output. This 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.

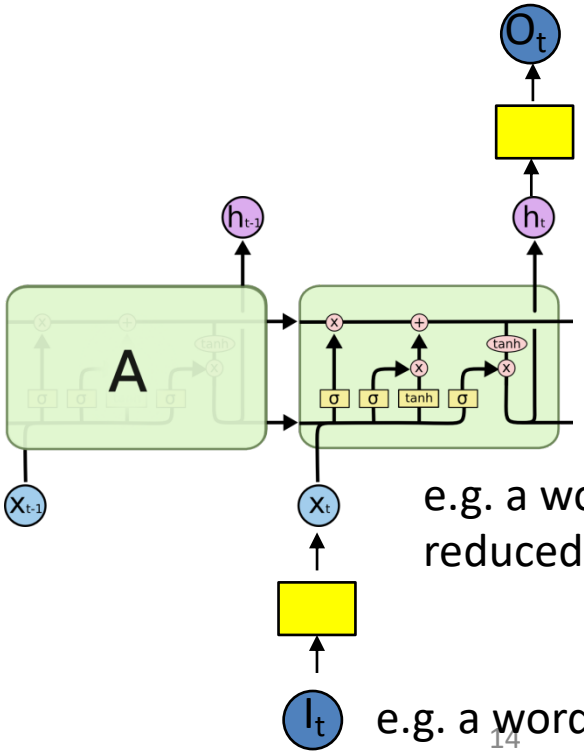


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

Overall Network Architecture

- Single or multilayer networks can compute LSTM inputs from problem inputs and problem outputs from LSTM outputs.



e.g. a POS tag as a "one hot" vector

e.g. a word "embedding" with reduced dimensionality

e.g. a word as a "one hot" vector

LSTM Training

- Trainable with backprop derivatives such as:
  - Stochastic gradient descent (randomize order of examples in each epoch) with momentum (bias weight changes to continue in same direction as last update).
  - ADAM optimizer (Kingma & Ma, 2015)

- Each cell has many parameters (W_f, W_i, W_C, W_o)
  - Generally requires lots of training data.
  - Requires lots of compute time that exploits GPU clusters.

General Problems Solved with LSTMs

- Sequence labeling
  - Train with supervised output at each time step computed using a single or multilayer network that maps the hidden state (h_t) to an output vector (O_t).
- Language modeling
  - Train to predict next input ($O_t = I_{t+1}$)
- Sequence (e.g. text) classification
  - Train a single or multilayer network that maps the final hidden state (h_n) to an output vector (O).

Summary of LSTM Application Architectures

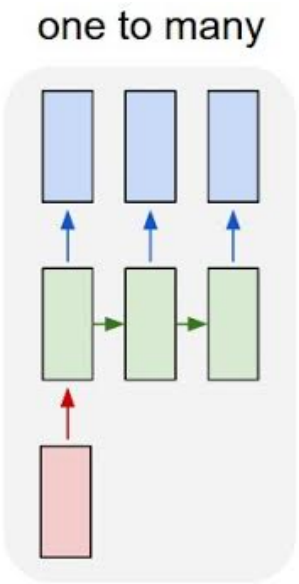
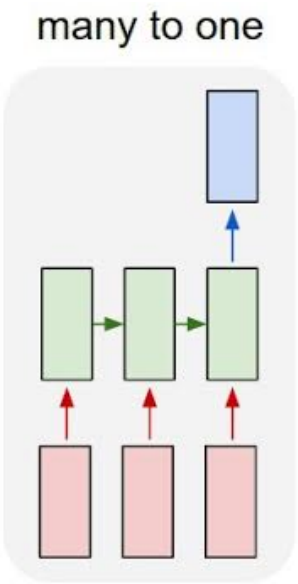
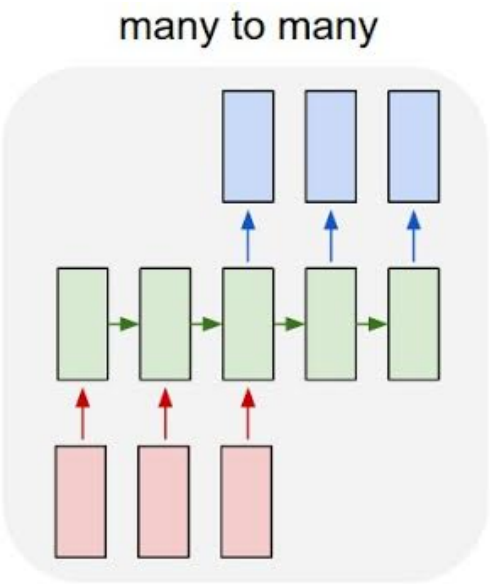


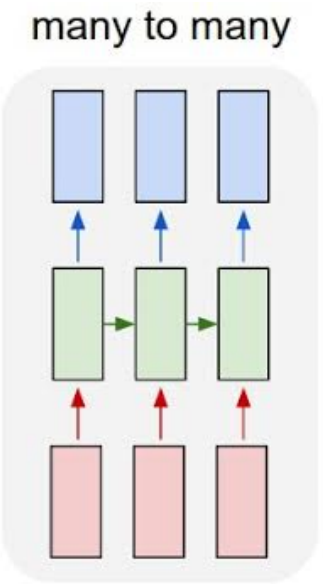
Image Captioning



Video Activity Recog
Text Classification



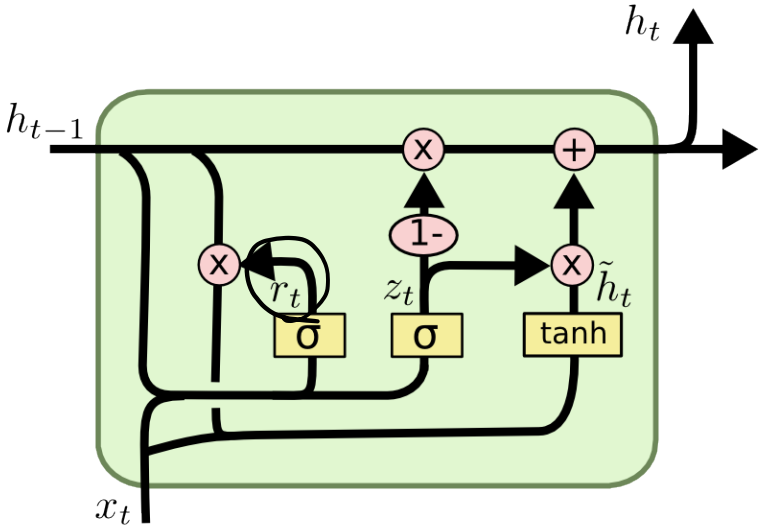
Video Captioning
Machine Translation



POS Tagging
Language Modeling

Gated Recurrent Unit(GRU)

- Alternative RNN to LSTM that uses fewer gates (Cho, et al., 2014)
  - Combines forget and input gates into “update” gate.
  - Eliminates cell state vector



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

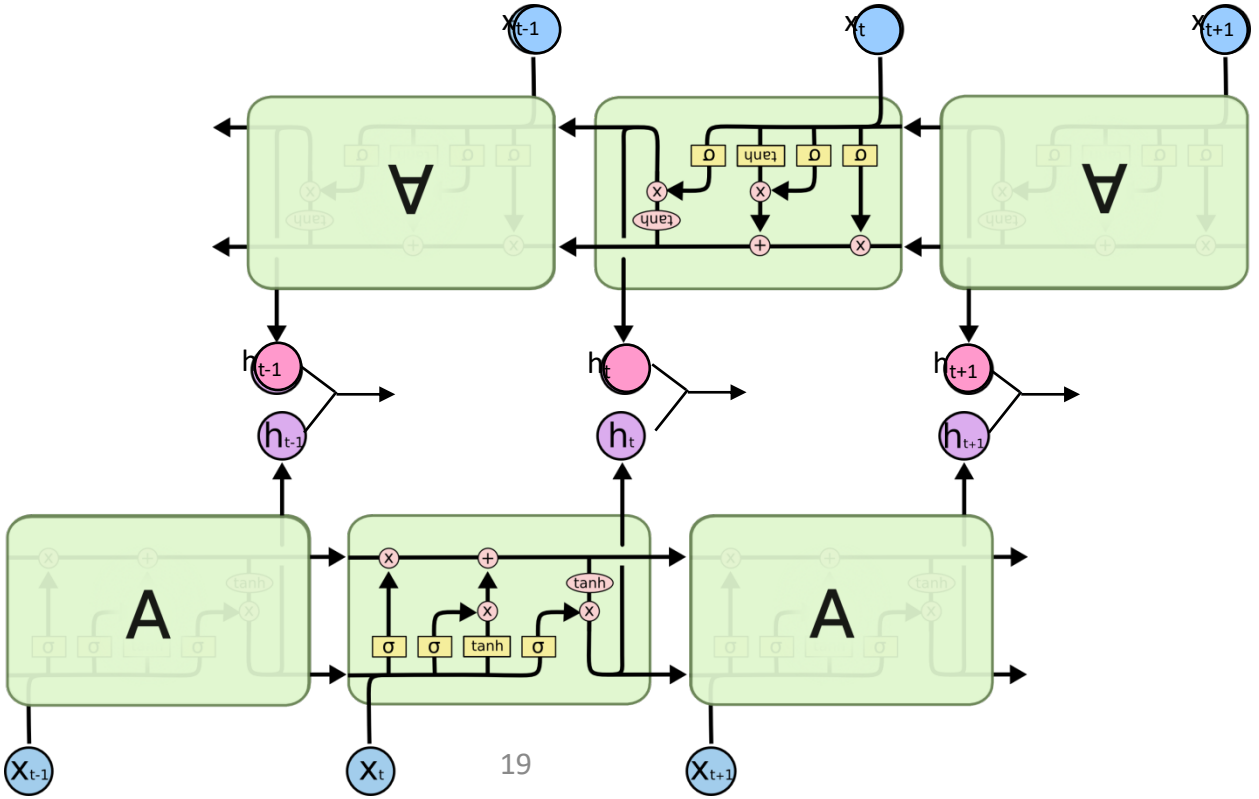
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

Bi-directional LSTM (Bi-LSTM)

- Separate LSTMs process sequence forward and backward and hidden layers at each time step are concatenated to form the cell output.



Successful Applications of LSTMs

- Speech recognition: Language and acoustic modeling
- Sequence labeling
  - POS Tagging
    https://www.aclweb.org/aclwiki/index.php?title=POS_Tagging_(State_of_the_art)
  - NER
  - Phrase Chunking
- Neural syntactic and semantic parsing
- Image captioning: CNN output vector to sequence
- Sequence to Sequence
  - Machine Translation (Sustkever, Vinyals, & Le, 2014)
  - Video Captioning (input sequence of CNN frame outputs)

GRU vs. LSTM

- GRU has significantly fewer parameters and trains faster.

- Experimental results comparing the two are still inconclusive, many problems they perform the same, but each has problems on which they work better.