LSTM Recurrent Neural Networks

Jiahui Chen
University of Arkansas
References: Wei MSU, colah github repo

Introduction to RNN

Recurrent neural networks (RNN) were developed by John Hopfield (1982). In 1993, an RNN used 1000 layers in time.

It is a very active research topic and one of the top ten methods in machine learning.

Motivation:

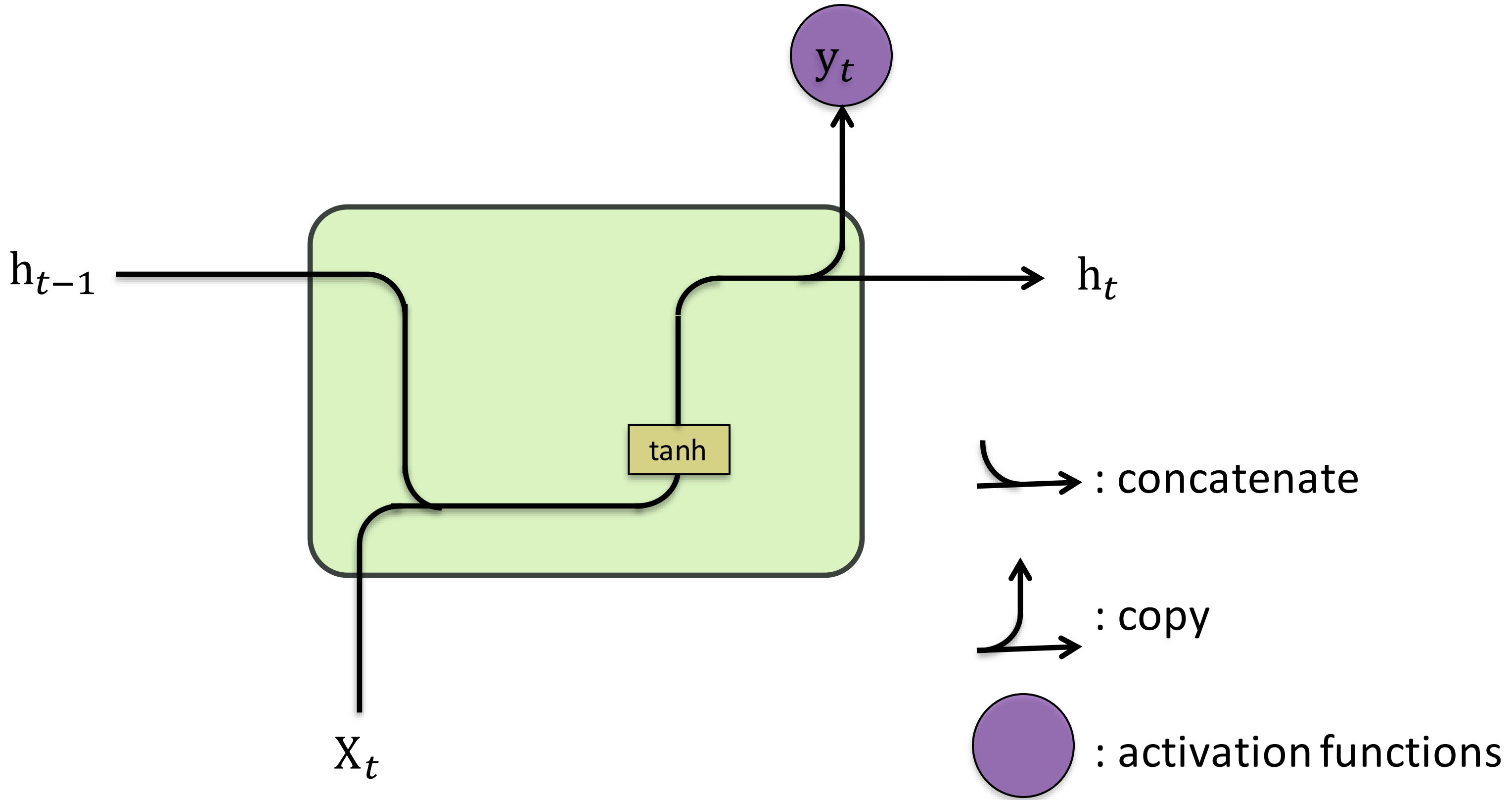
Temporal patterns or correlations in sequential data, such as speech recognition, handwriting recognition, credit card crime prevention, drug optimization, subject of a researcher’s next paper etc., can be used to make better predictions.

Introduction to RNN

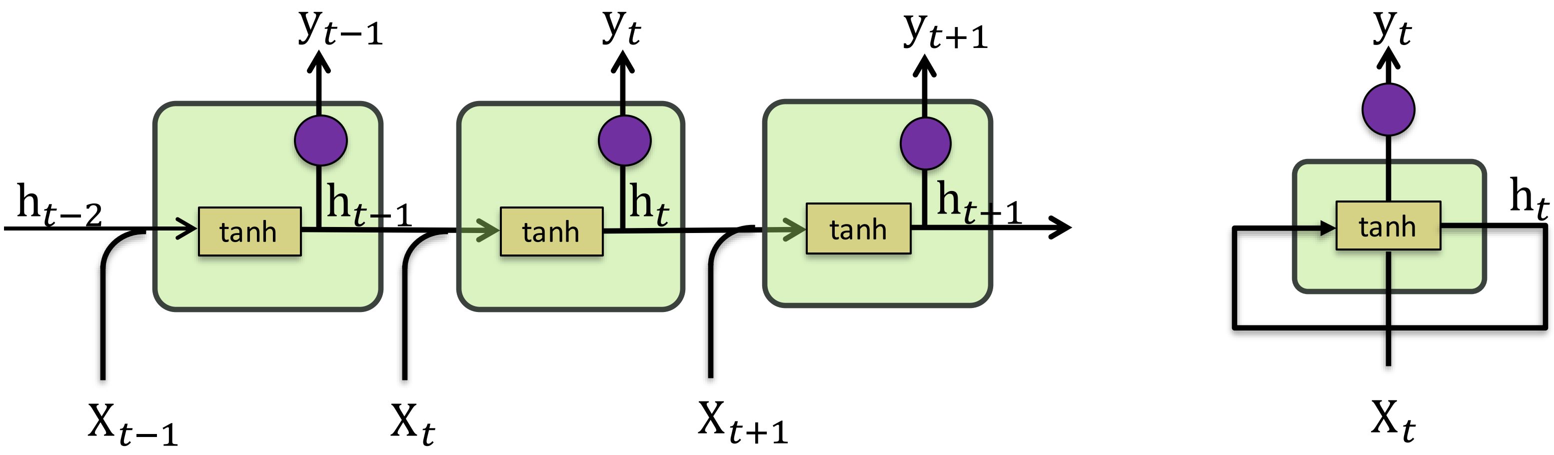
Sequence modeling of sequential data: Design objective

- Deal with variable-length sequences
- Track long-term dependencies
- Maintain information about the order
- Share parameters across the sequence

A memory cell of regular RNN



Simple RNN – Elman Network



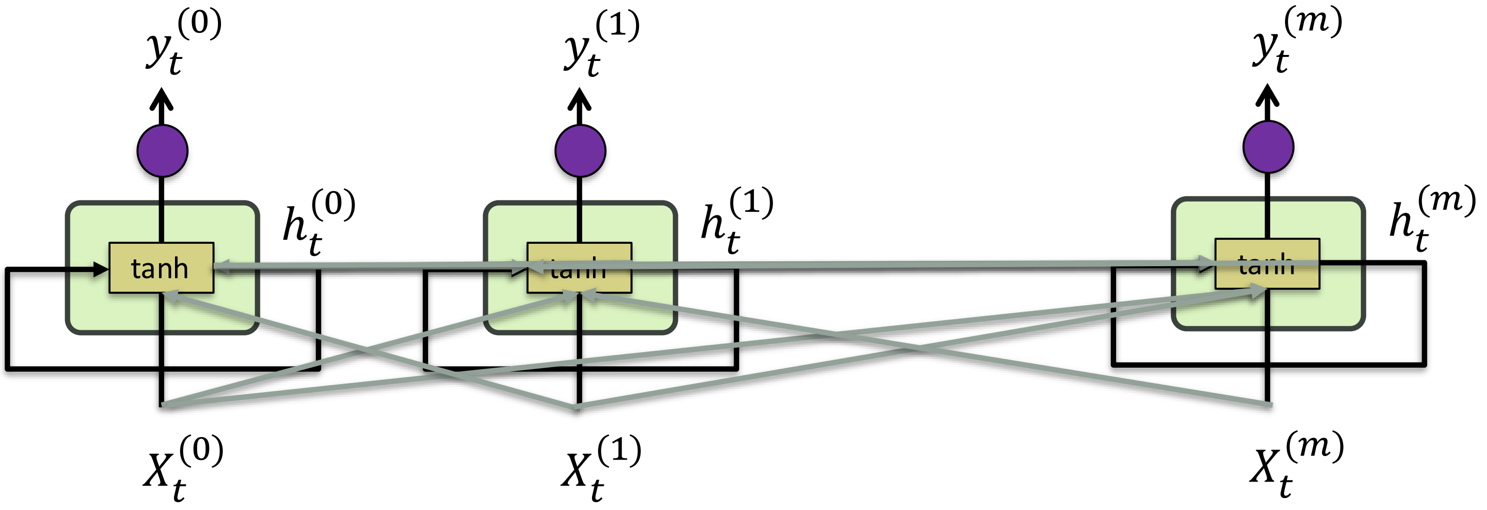
$$h_t = \sigma_h(C_h X_t + U h_{t-1} + b_h)$$

$$\hat{y}_t = \sigma_y(C_y h_t + b_y)$$

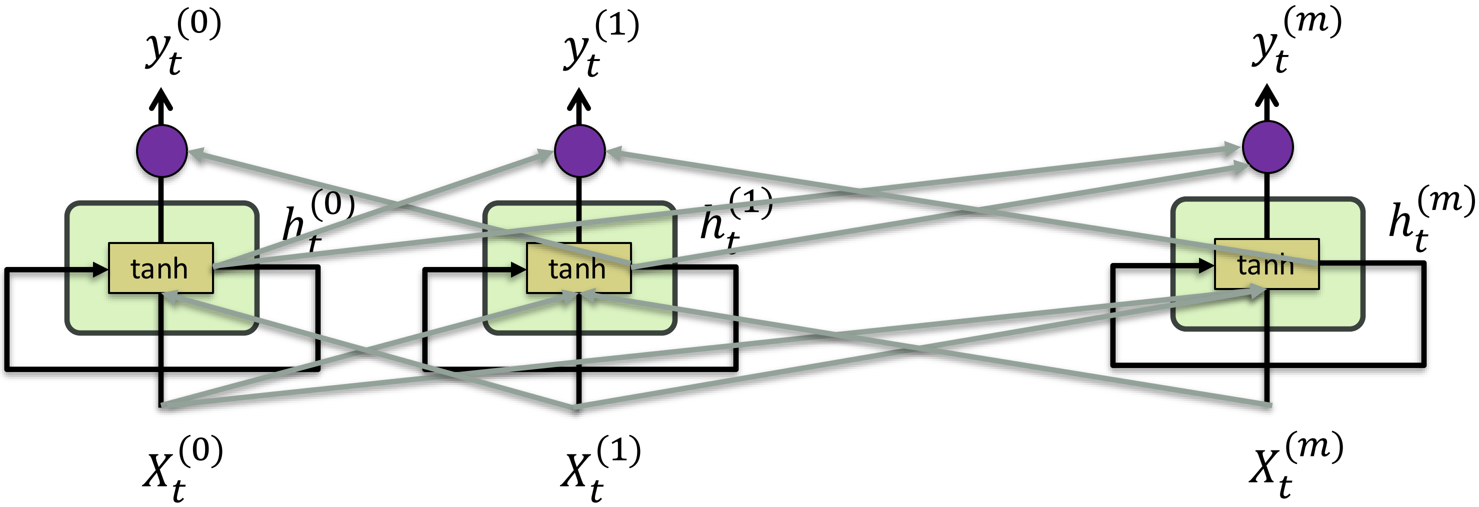
Weight: C_h, U

Hidden: h_t

Simple RNN – Elman Network



Simple RNN – Jordan Network

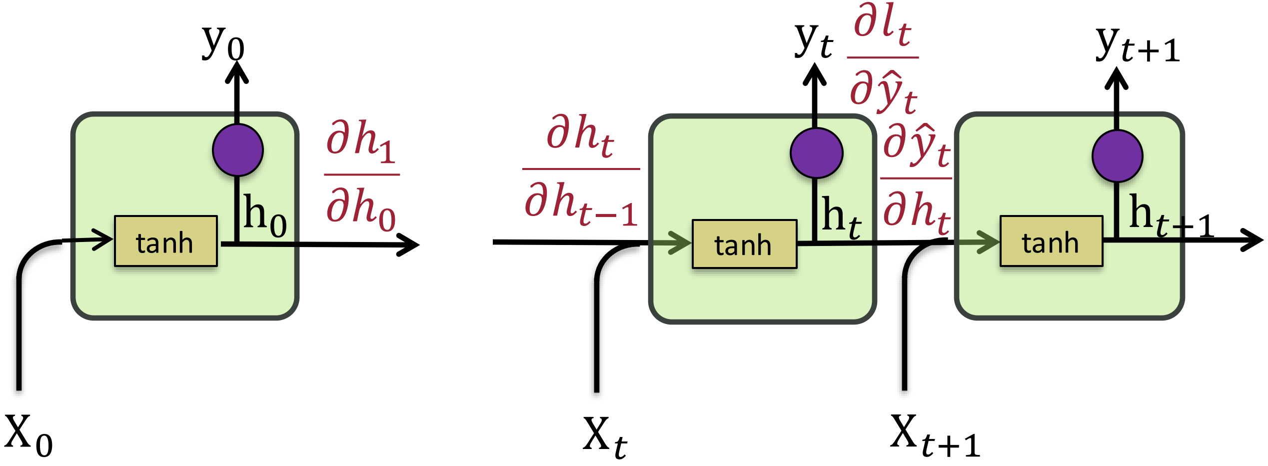


Loss Function of RNN

- $h_t = \sigma_h(C_h X_t + U h_{t-1} + b_h)$ tanh is used
- $\hat{y}_t = \sigma_y(C_y h_t + b_y)$ softmax is used
- Cross entropy loss function for each step
- $l_t(\hat{y}_t, y_t) = -\sum_i^M y_t^{(i)} \log \hat{y}_t^{(i)}$
- Accumulated total error
- $L(\hat{y}_t, y_t) = \sum_t l_t(\hat{y}_t, y_t)$

Backpropagation

- Chain rule: $\frac{\partial L}{\partial C_{ij}} = \sum_t \frac{\partial l_t(\hat{y}_t, y_t)}{\partial C_{ij}}$
- $\frac{\partial l_t(\hat{y}_t, y_t)}{\partial C_{ij}} = \sum_{k=0}^{t} \frac{\partial l_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial C_{ij}}$



Vanishing Gradient Issue

- $\frac{\partial l_t(\hat{y}_t, y_t)}{\partial C_{ij}} = \sum_{k=0}^{t} \frac{\partial l_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \prod_{m=k+1}^{t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial C_{ij}}$
- Gradients from far away steps vanish. As a result, RNN does not maintain the long-range memory, which is needed in many situations.

Vanishing Gradient Issue

Solutions

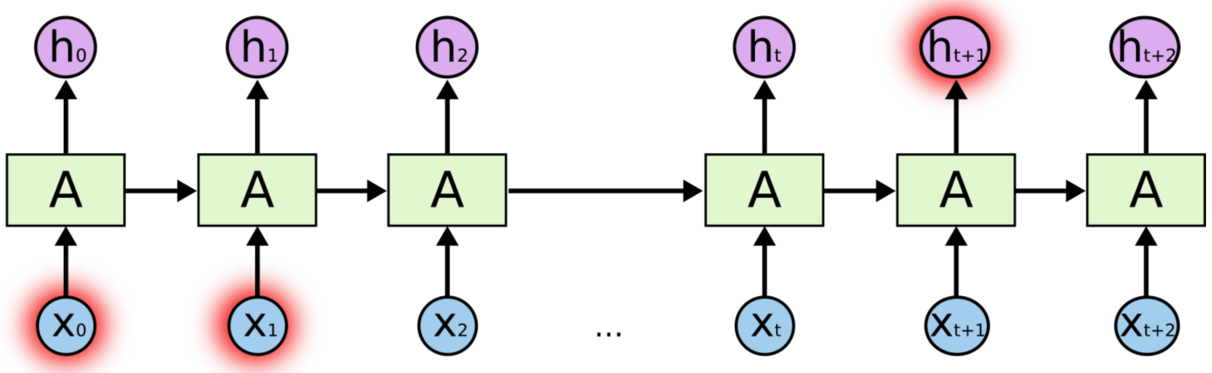
- Use appropriate treatment of weight matrices: initialization and normalization
- Use ReLu, which is linear for positive values
- Use Long-Short-Term Memory
- Use Gated Recurrent Units
- Use Residual network (ResNets)
- Use Deep belief network, etc.

Vanishing/Exploding Gradient Problem

- Backpropagated errors multiply at each layer, resulting in exponential decay (if derivative is small) or growth (if derivative is large).
- Makes it very difficult train deep networks, or simple recurrent networks over many time steps.

Long Distance Dependencies

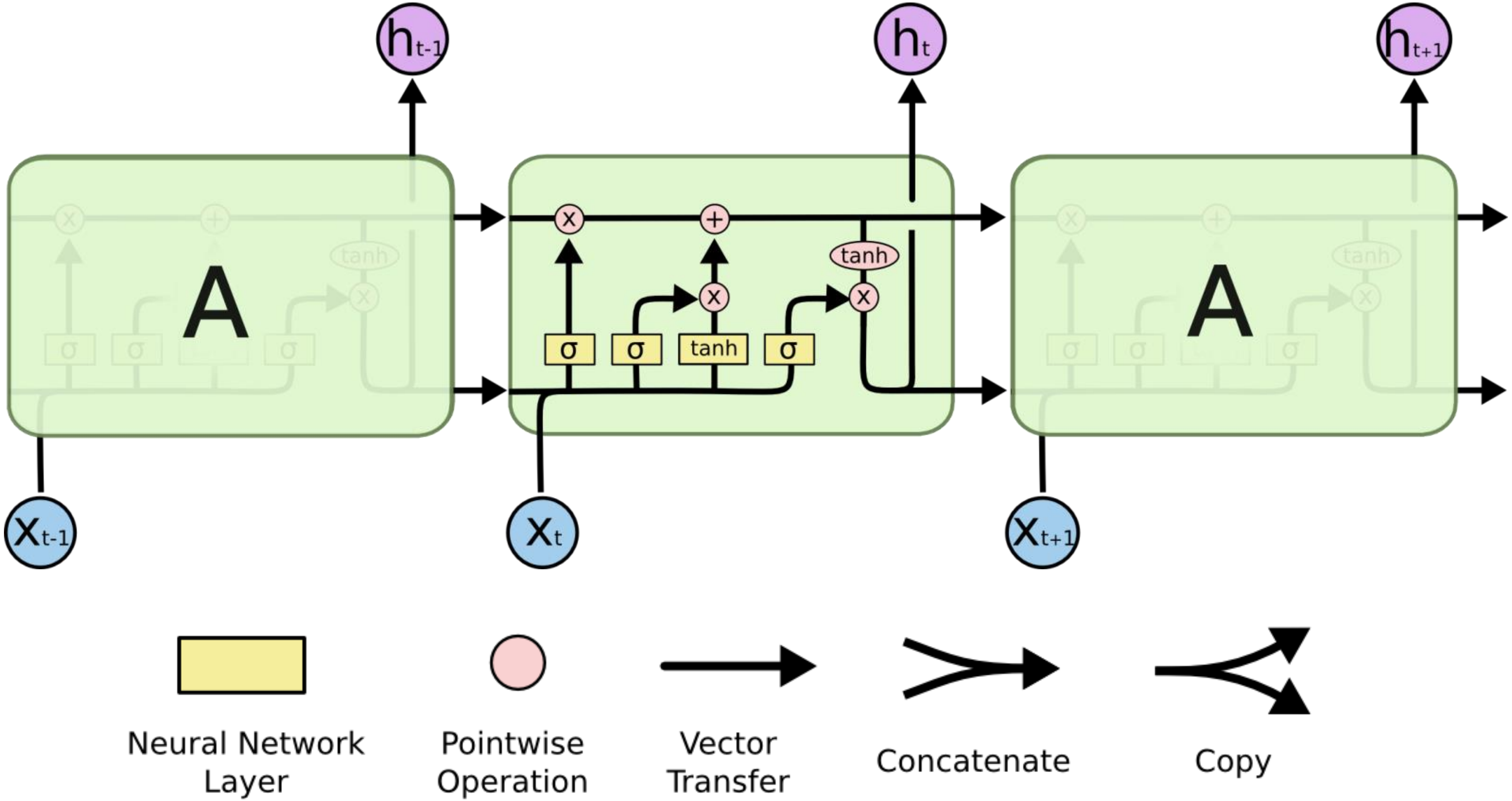
- It is very difficult to train RNNs to retain information over many time steps
- This make is very difficult to learn RNNs that handle long-distance dependencies, such as subject-verb agreement.



Long Short Term Memory

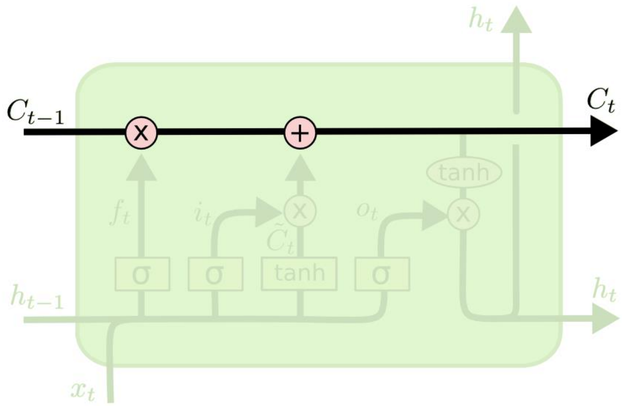
- LSTM networks, add additional gating units in each memory cell.
  - Forget gate
  - Input gate
  - Output gate
- Prevents vanishing/exploding gradient problem and allows network to retain state information over longer periods of time.

LSTM Network Architecture



Cell State

- Maintains a vector C_t that is the same dimensionality as the hidden state, h_t
- Information can be added or deleted from this state vector via the forget and input gates.

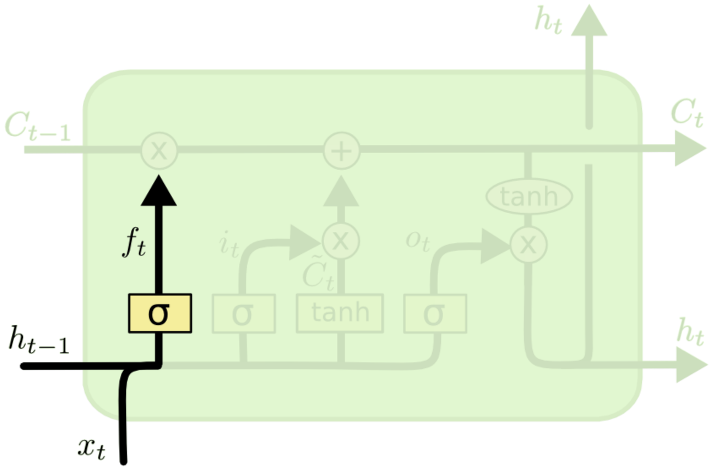


Cell State Example

- Want to remember person & number of a subject noun so that it can be checked to agree with the person & number of verb when it is eventually encountered.
- Forget gate will remove existing information of a prior subject when a new one is encountered.
- Input gate "adds" in the information for the new subject.

Forget Gate

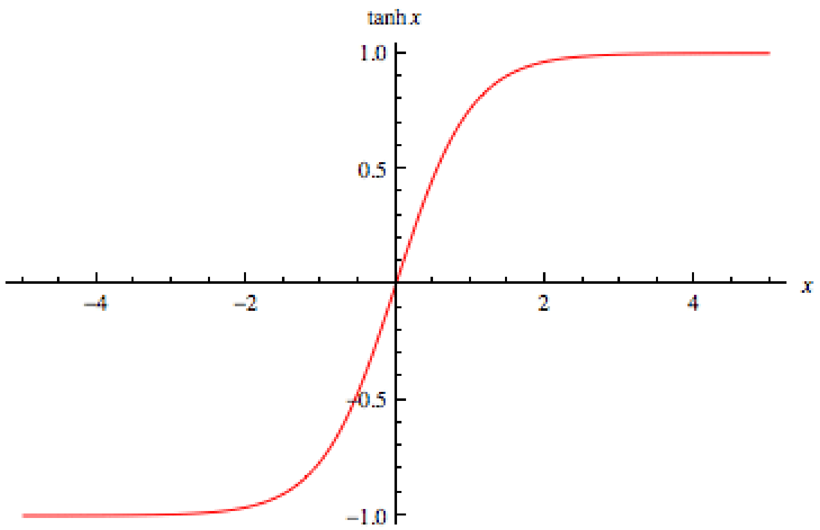
- Forget gate computes a 0-1 value using a logistic sigmoid output function from the input, x_t, and the current hidden state, h_t:
- Multiplicatively combined with cell state, "forgetting" information where the gate outputs something close to 0.



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

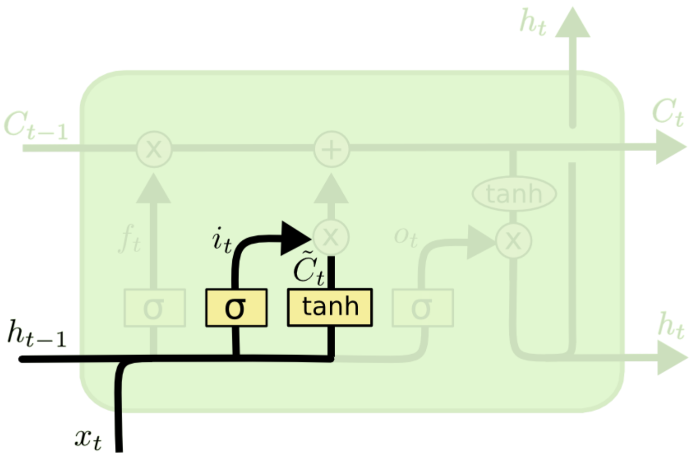
Hyperbolic Tangent Units

- 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.



Input Gate

- First, determine which entries in the cell state to update by computing 0-1 sigmoid output.
- Then determine what amount to add/subtract from these entries by computing a tanh output (valued –1 to 1) function of the input and hidden state.

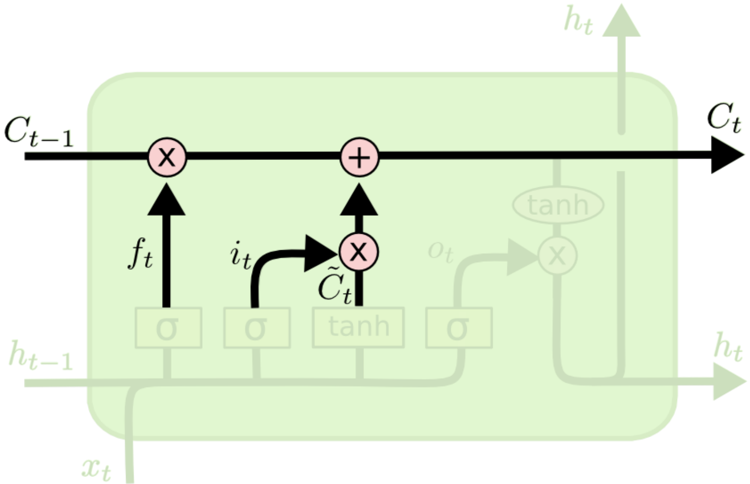


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

Updating the Cell State

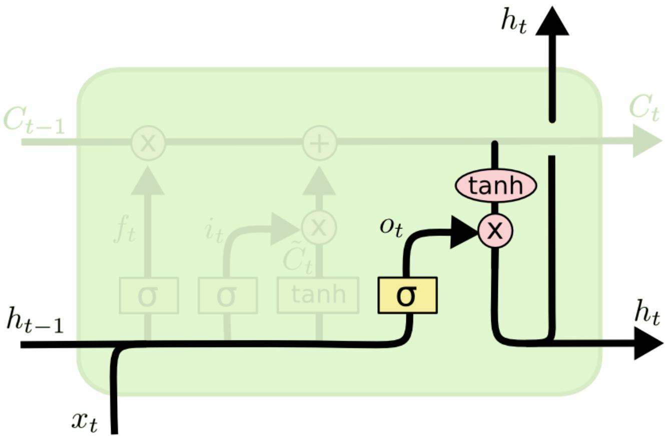
Cell state is updated by using component-wise vector multiply to "forget" and vector addition to "input" new information.



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

Output Gate

- Hidden state is updated based on a "filtered" version of the cell state, scaled to –1 to 1 using tanh.
- Output gate computes a sigmoid function of the input and current hidden state to determine which elements of the cell state to "output".



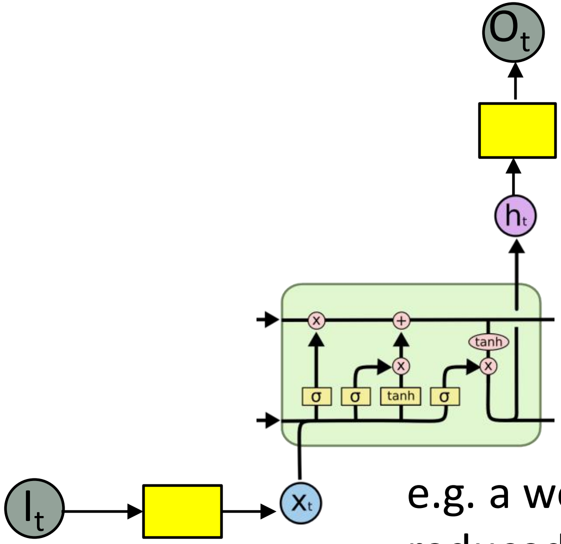
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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

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 as a "one hot" vector

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

LSTM Training

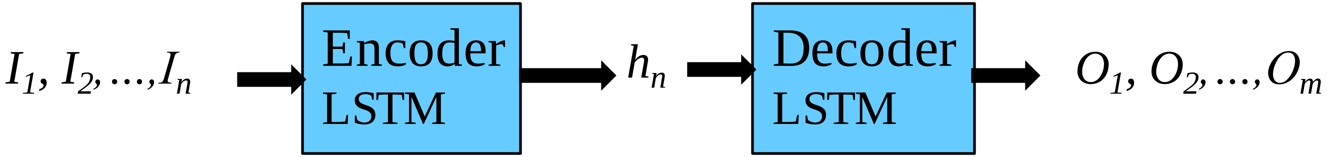
- Trainable with backprop derivatives such as:
  - Stochastic gradient descent (randomize order of examples in each epoch) with momentum (bias weight changes to continue in same direction as last update).
  - ADAM optimizer (Kingma & Ma, 2015)
- Each cell has many parameters (W_f, W_i, W_C, W_o)
  -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

Sequence to Sequence Transduction

- Encoder/Decoder framework maps one sequence to a "deep vector" then another LSTM maps this vector to an output sequence.



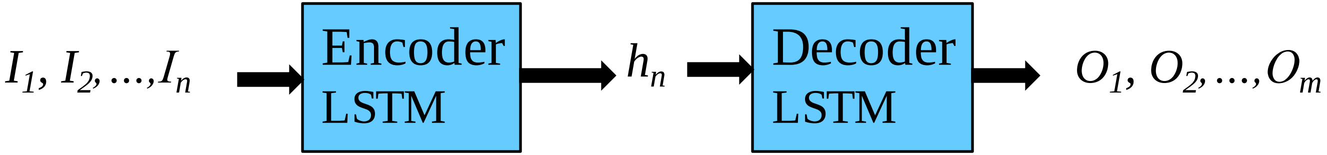
- Train model "end to end" on I/O pairs of sequences.

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).

Sequence to Sequence Transduction

- Encoder/Decoder framework maps one sequence to a "deep vector" then another LSTM maps this vector to an output sequence.



- Train model "end to end" on I/O pairs of sequences.

LSTM Application Architectures

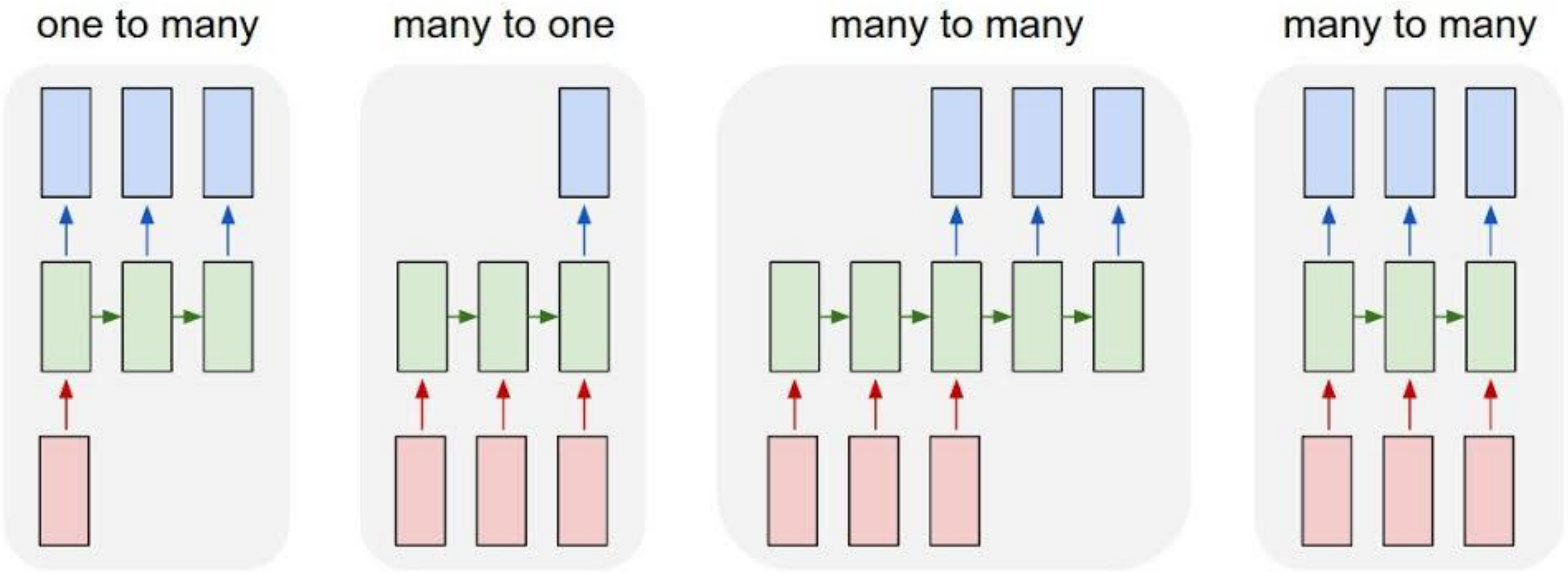


Image Captioning

Video Activity Recog
Text Classification

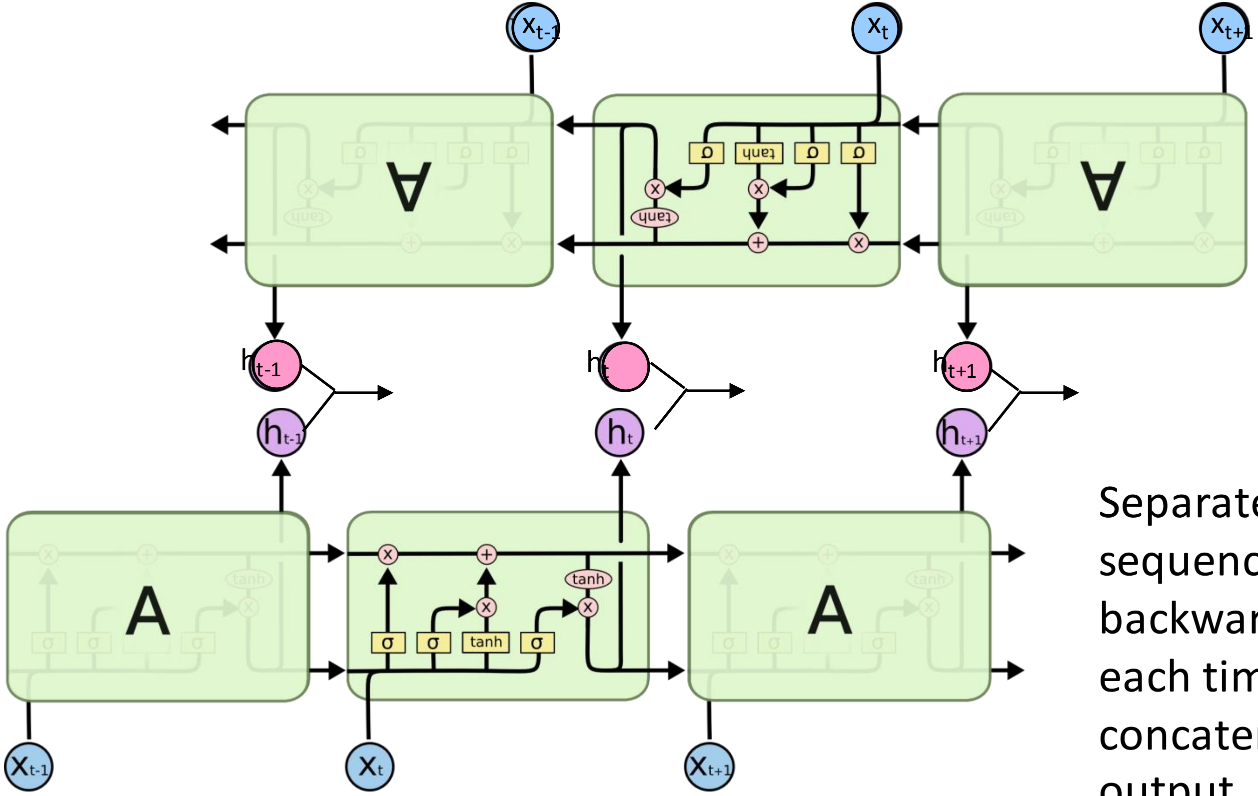
Video Captioning
Machine Translation

POS Tagging
Language Modeling

Successful Applications of LSTM

- 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)

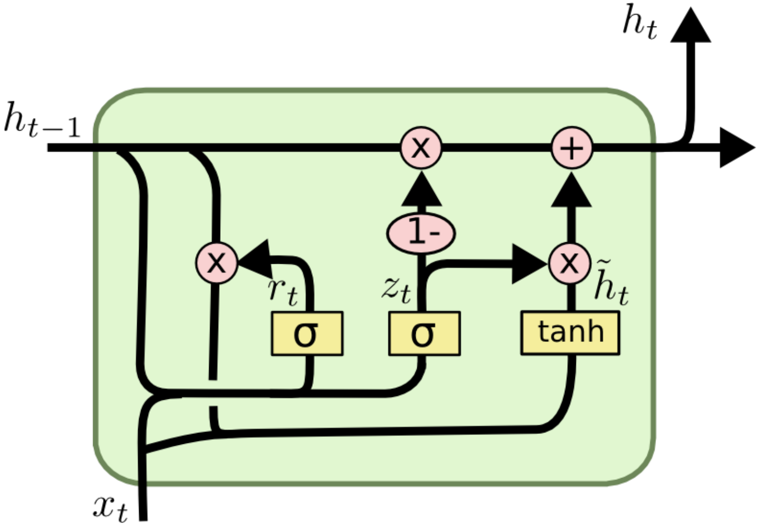
Bi-directional LSTM (Bi-LSTM)



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

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



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

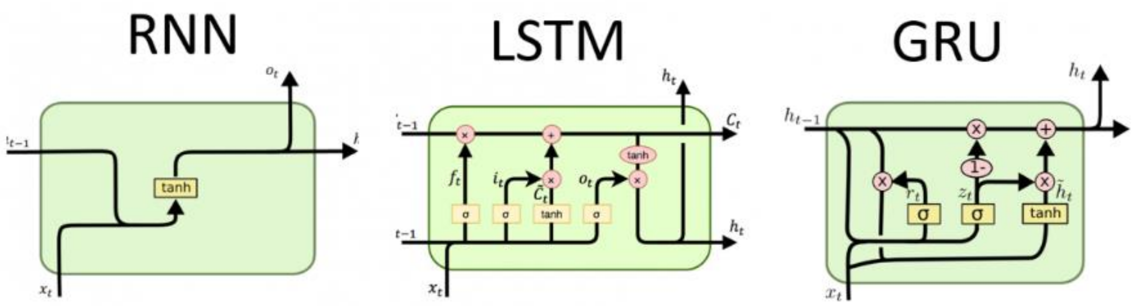
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

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.



Attention

- For many applications, it helps to add “attention” to RNNs.
- Allows network to learn to attend to different parts of the input at different time steps, shifting its attention to focus on different aspects during its processing.
- Used in image captioning to focus on different parts of an image when generating different parts of the output sentence.
- In MT, allows focusing attention on different parts of the source sentence when generating different parts of the translation.

Conclusions

- By adding "gates" to an RNN, we can prevent the vanishing/exploding gradient problem.
- Trained LSTMs/GRUs can retain state information longer and handle long-distance dependencies.
- Recent impressive results on a range of challenging NLP problems.