Large Scale Data Analysis Using Deep Learning

Sequence Modeling: Recurrent and Recursive Nets

U Kang
Seoul National University

In This Lecture

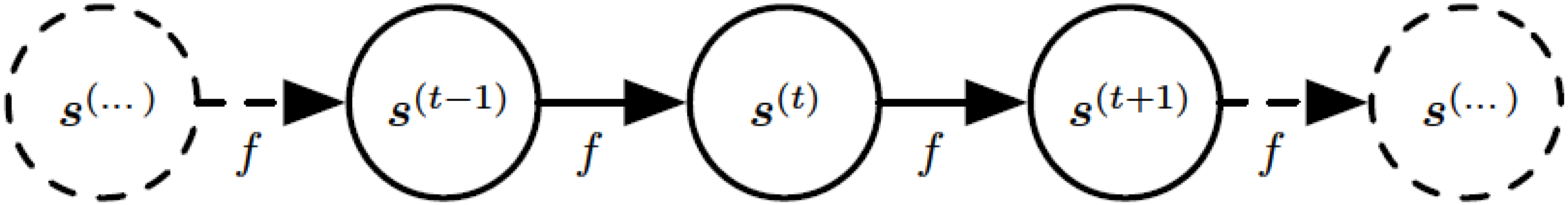
- Recurrent Neural Network
  - Main idea
  - Major architectures
  - Problem of long-term dependencies and how to solve them (LSTM, etc.)

RNN

- Recurrent neural network (RNN)
  - A family of neural networks for processing sequential data
  - Can scale to much longer sequences than other networks do
  - Can process sequences of variable (or infinite) length
- To go from multi-layer networks to RNN
  - Sharing parameters across different parts of a model
    - Allows extending the model to examples of different length
    - Important when a specific piece of information can occur at multiple positions within the sequence
    - E.g., recognize year 2009 as the relevant piece of information in the two sentences "I went to Nepal in 2009" and "In 2009, I went to Nepal"

Classical Dynamical System

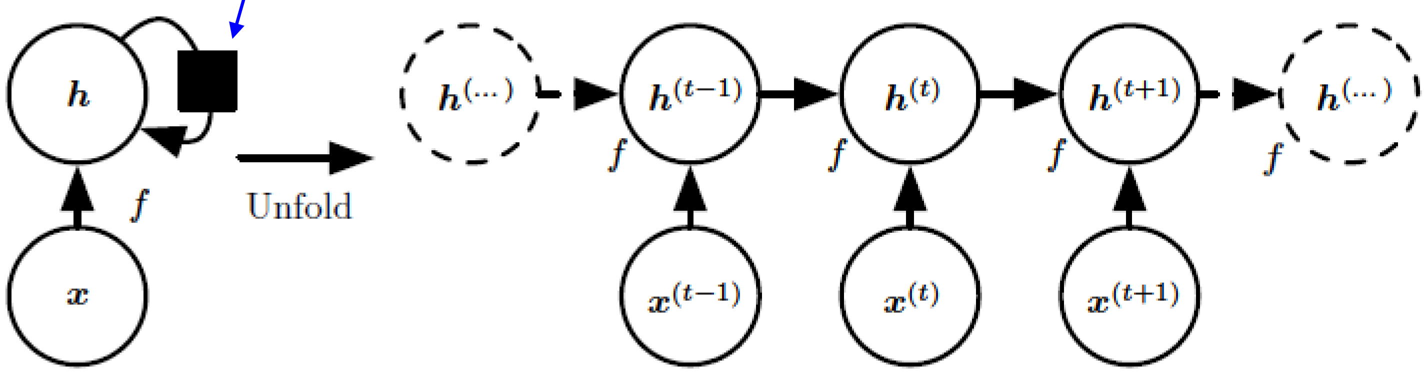
- Consider the classical form of a dynamical system: $s^{(t)} = f(s^{(t-1)};\ \theta)$
- The system can be expressed with the unfolded computational graph



Unfolding Computation Graphs

- Consider a dynamical system driven by an external signal $x^{(t)}$
  - $h^{(t)} = f(h^{(t-1)}, x^{(t)};\ \theta)$

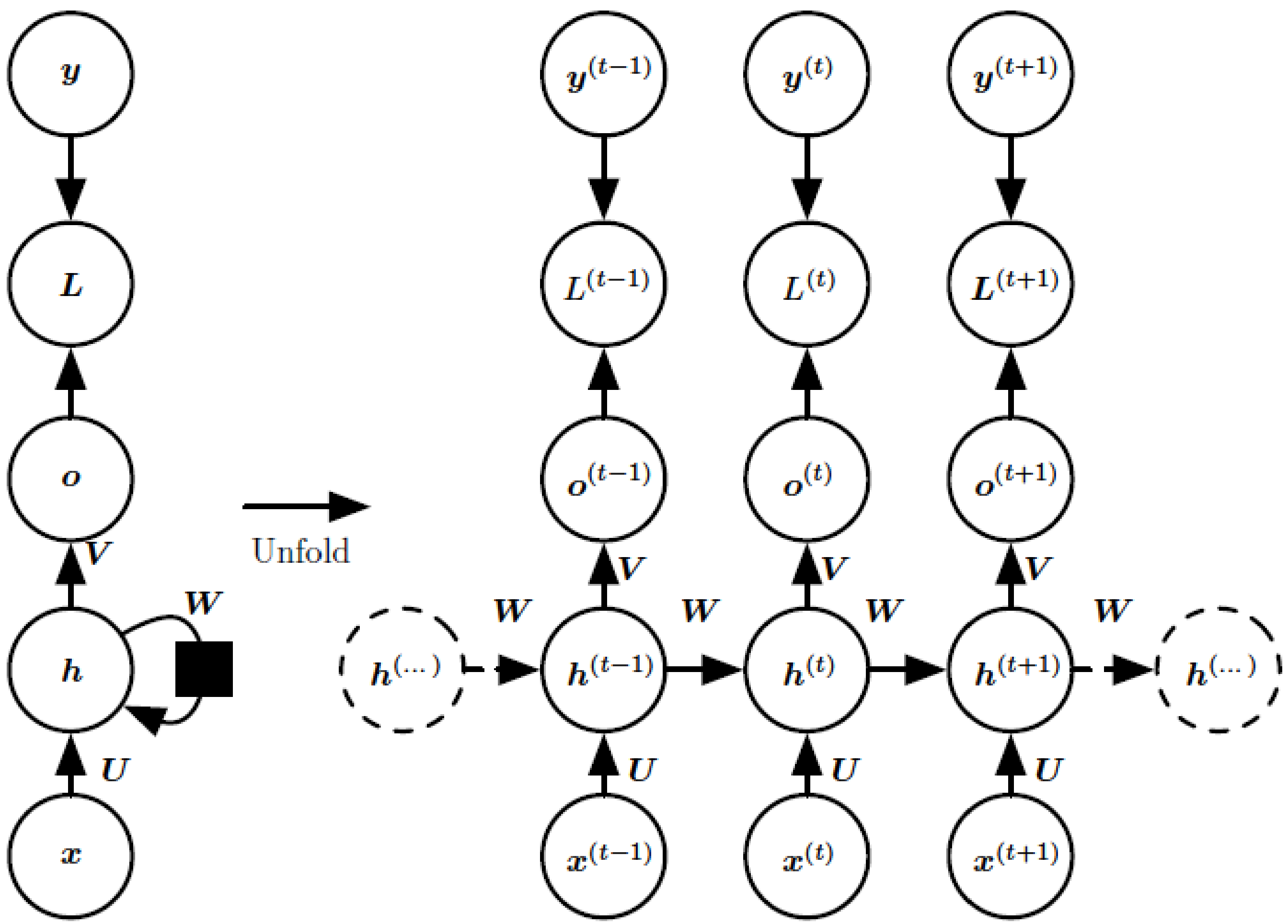
delay of a single
time step



recurrent graph
or circuit diagram

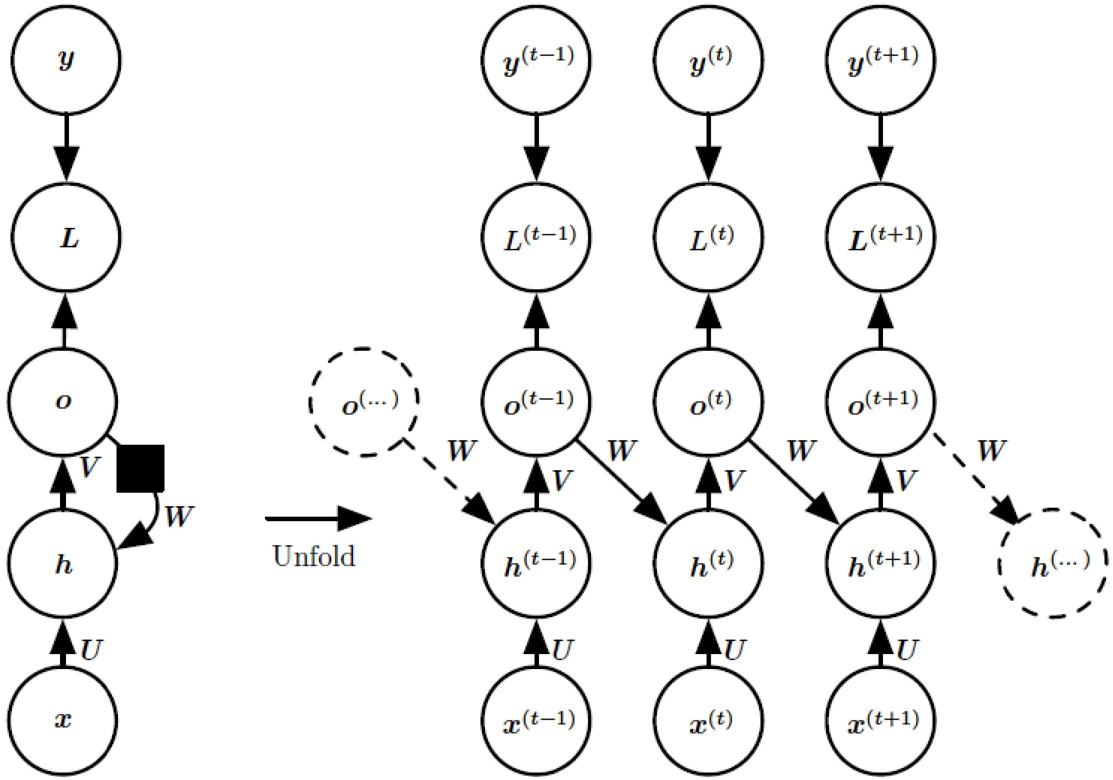
unrolled graph

Recurrent Hidden Units



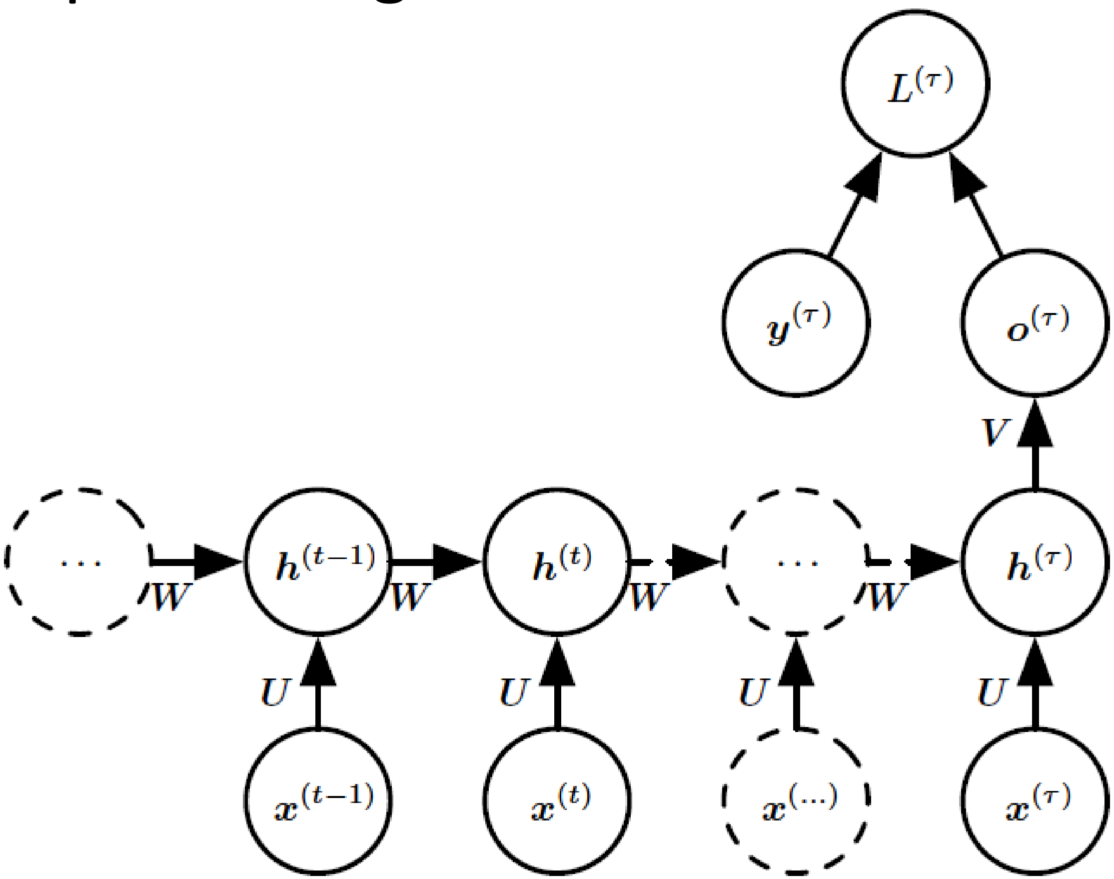
Recurrent through only the Output

- Less powerful than the previous model since the output cannot encode all the information in the hidden node
- But, it allows efficient training since each time step can be trained in isolation from the others (will be described soon)



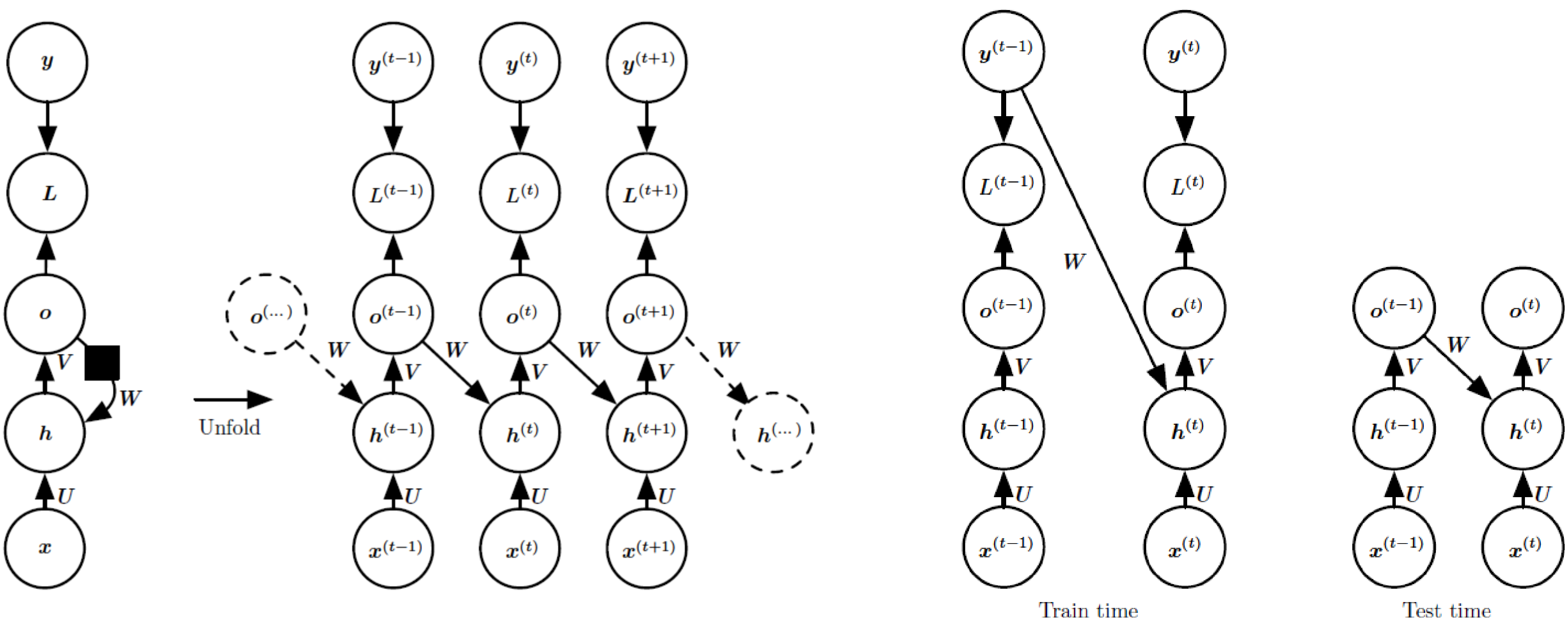
Sequence Input, Single Output

- Used to summarize a sequence and produce a fixed-size representation used as input for further processing



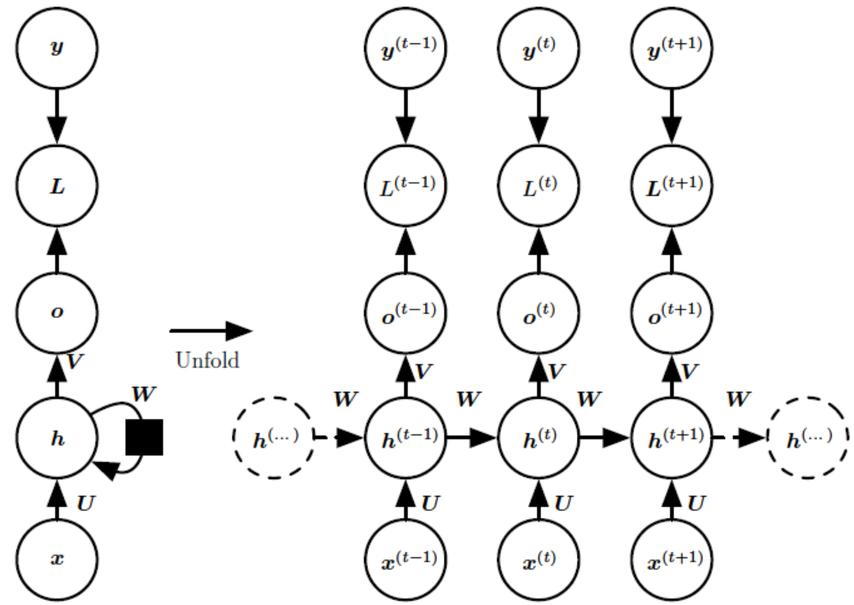
Teacher Forcing

- An RNN, where recurrent connections are from the output at one time step to the hidden units at the next time step, can be trained efficiently with teacher forcing
  - Enables parallel learning



Forward/Back Propagation in RNN

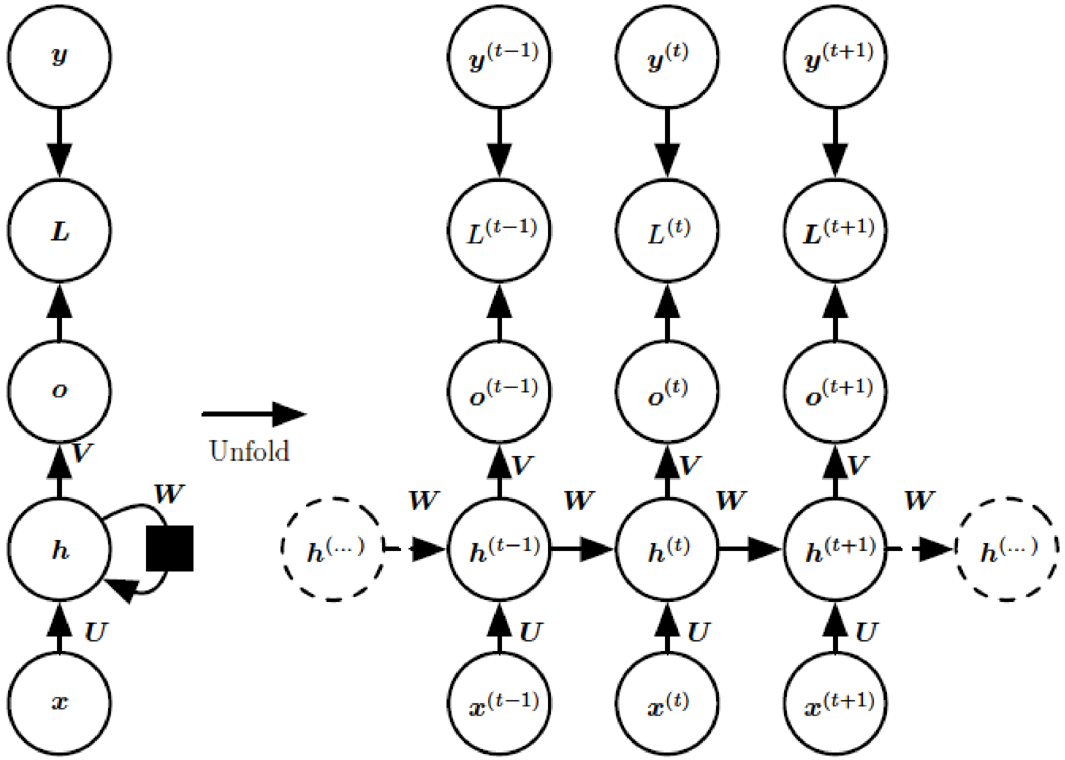
- $\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$
- $\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)})$
- $\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}$
- $\hat{\boldsymbol{y}}^{(t)} = softmax(\boldsymbol{o}^{(t)})$
- The total loss is the sum of the losses over all time steps:
  - $L(\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \dots, \boldsymbol{y}^{(\tau)}\})$
    $= \sum_t L^{(t)}$
    $= -\sum_t \log p_{model}(\boldsymbol{y}^{(t)} | \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(\tau)}\})$



- Use back propagation through time (BPTT) to compute gradient
  - BPTT is essentially the same standard back-propagation algorithm on the unfolded computational graph

Modeling Sequences Conditioned on Context

- The RNN in the figure below models $P(x;\theta)$, where y's are used only to evaluate the model
- We can also use RNN to model $P(y|x)$, by using $P(y|w)$ where $w = f(x;\theta)$ is a function of x.



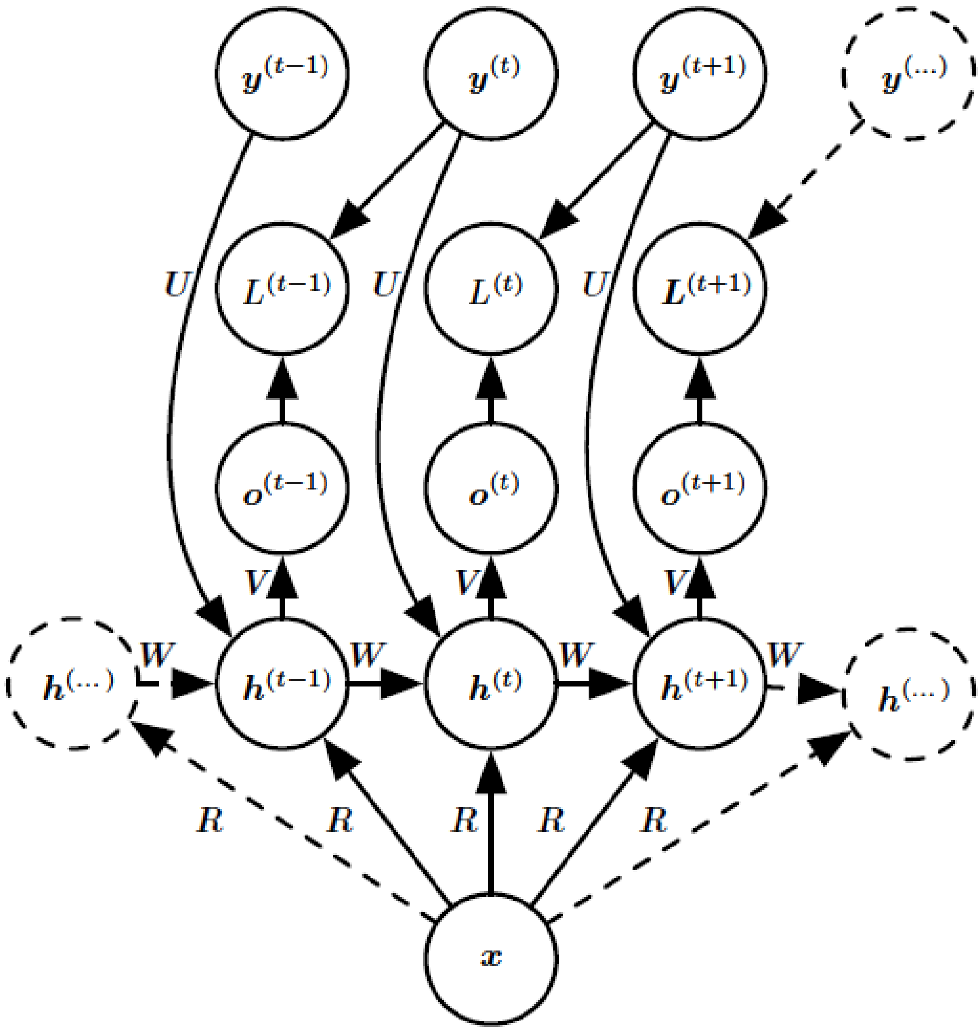
Modeling Sequences Conditioned on Context

- Modeling $P(y^{(t)}|x)$ for a fixed x: make it an extra input of the RNN that generates the y sequence

- How to provide an extra input to an RNN?
  - Add the input as an extra input at each time step
  - Add the input as the initial state $h^{(0)}$, or
  - both

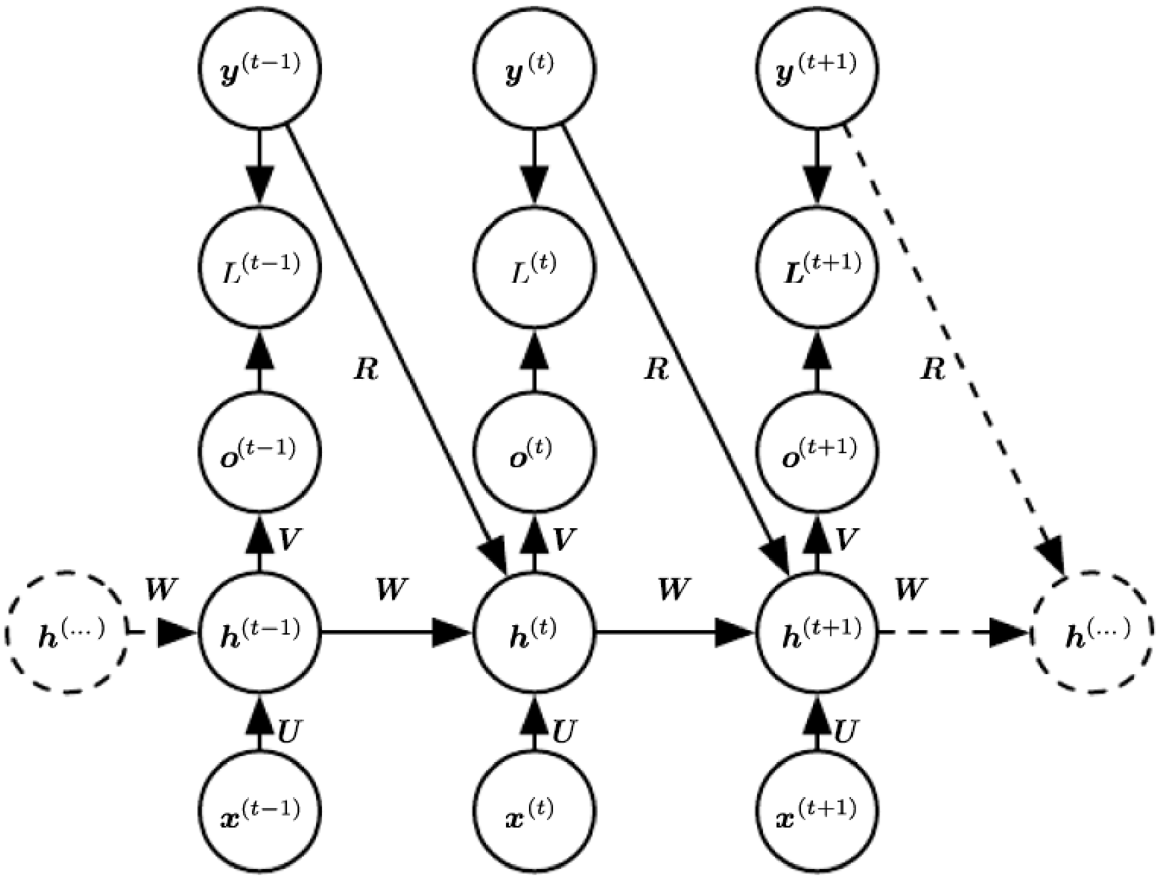
Vector to Sequence

- Adding an extra input x at each time step



Hidden and Output Recurrence

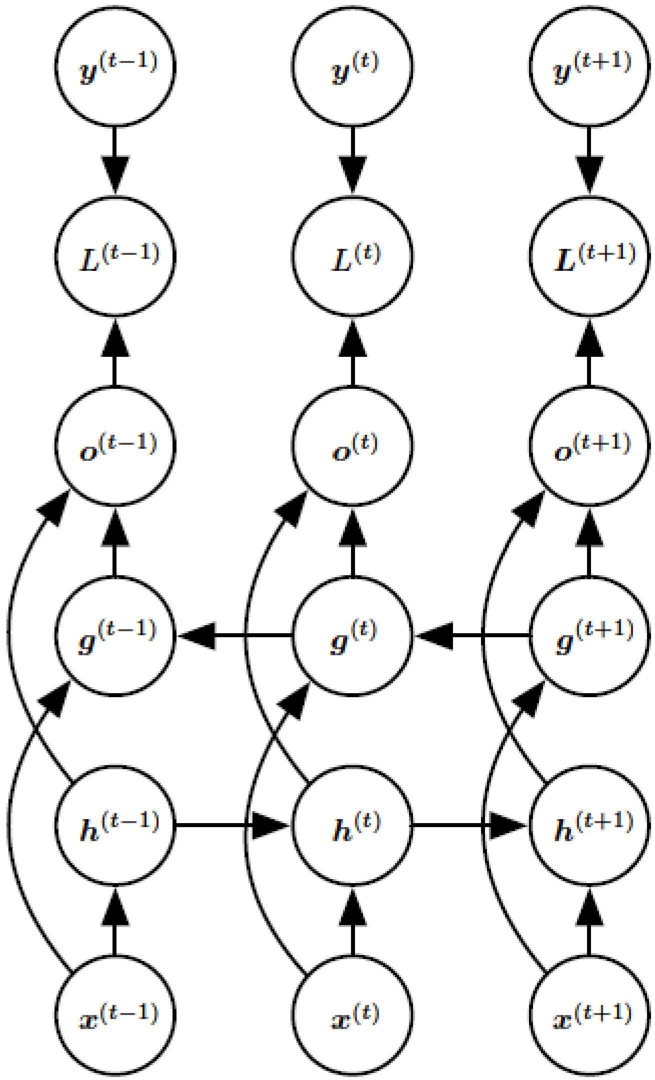
- RNN may receive a sequence of vectors $x^{(t)}$ as extra input



Bidirectional RNN

- All of the RNN we have considered up to now have a "causal" structure
  - I.e., the state at time t only captures information from the past, $x^{(1)}, \ldots, x^{(t-1)}$, and the present input $x^{(t)}$
- However, in many applications we want to output prediction of $y^{(t)}$ which may depend on the whole input sequence
  - Speech recognition: the correct interpretation as a phoneme of the current sound may depend on the next few phonemes
  - Handwriting recognition
  - Bioinformatics
- Bidirectional RNNs were invented to address that need

Bidirectional RNN

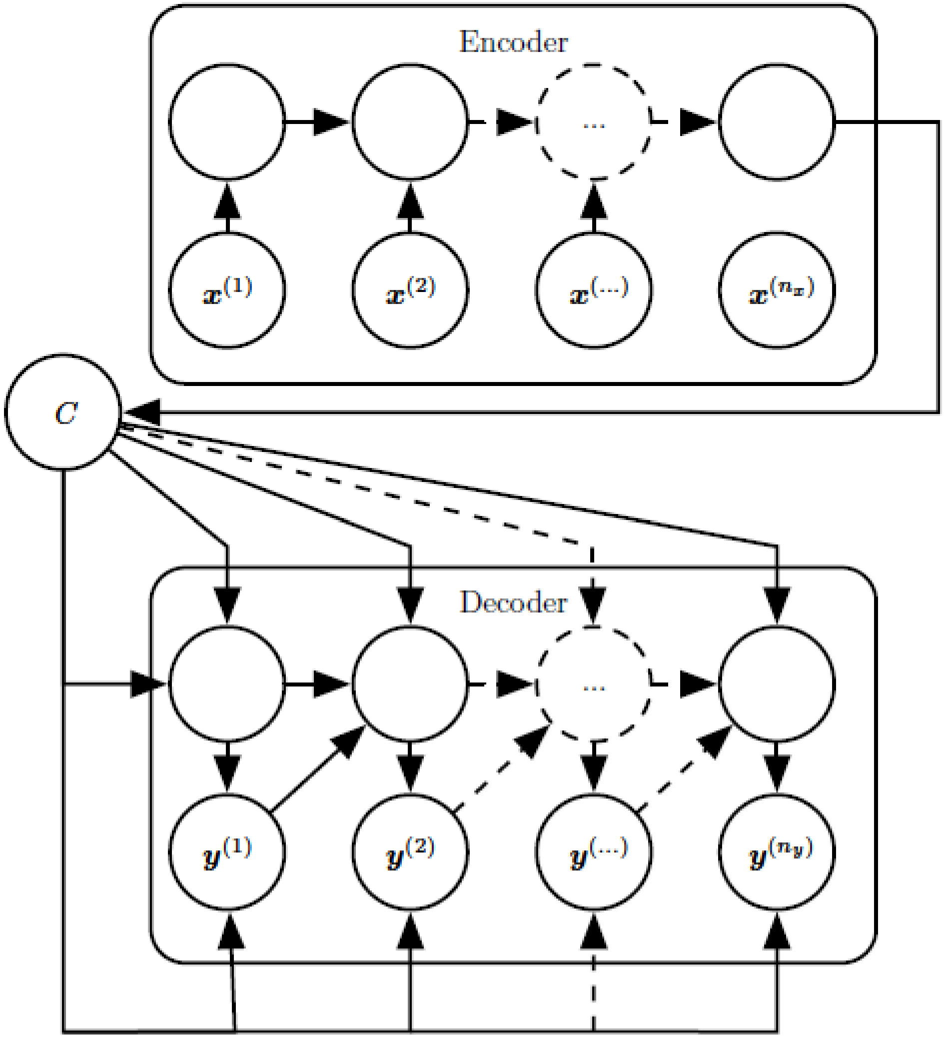


Sequence to Sequence Architecture

- Training RNN to map an input sequence to an output sequence which is not necessarily of the same length
  - Speech recognition
  - Machine translation
  - Question answering

- The simplest RNN architecture for mapping a variable-length sequence to another variable-length sequence is called *sequence-to-sequence* or *encoder-decoder* architecture

Sequence to Sequence Architecture



Sequence to Sequence Architecture

- *Sequence-to-sequence* or *encoder-decoder* architecture
  - An encoder or reader or input RNN processes the input sequence $X = (x^{(1)}, \dots, x^{n_x})$, and emits the context C, usually as a simple function of its final hidden state
  - A decoder or writer or output RNN is conditioned on that fixed-length vector to generate the output sequence $Y = (y^{(1)}, \dots, y^{(n_y)})$
  - Note that n_x and n_y can be different
  - The two RNNs are trained jointly to maximize the average of $\log P(y^{(1)}, \dots, y^{(n_y)} | x^{(1)}, \dots, x^{n_x})$ over all the pairs of x and y sequences in the training set
  - The last state h_{n_x} of the encoder RNN is typically used as a representation C of the input sequence that is provided as input to the decoder RNN

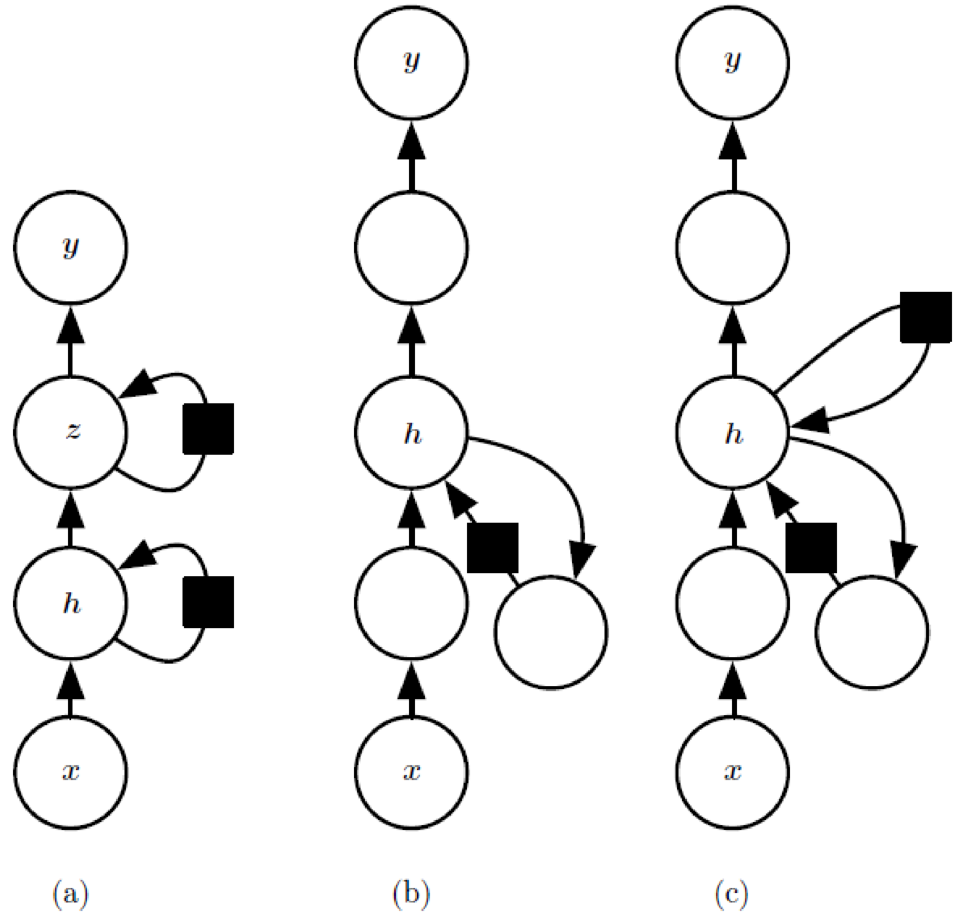
Deep RNNs

- Computation in most RNNs can be decomposed into three blocks of parameters and associated transformations
  - From the input to the hidden state
  - From the previous hidden state to the next hidden state
  - From the hidden state to the output

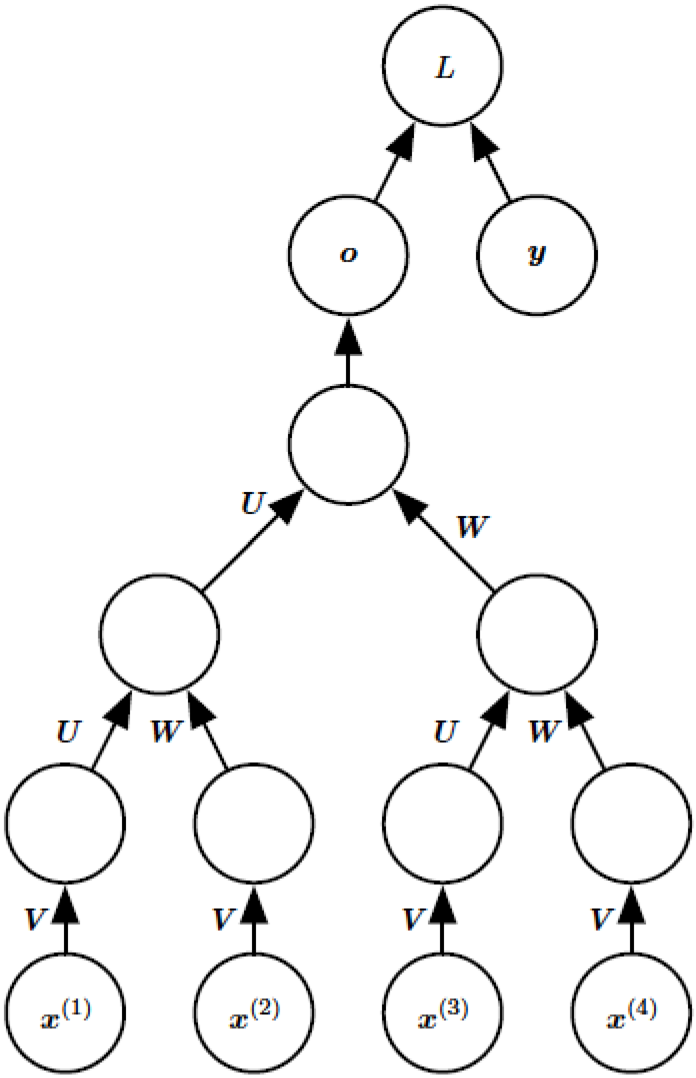
- Deep RNN: introduce depth in each of these operations

Deep RNNs

- (a) two hidden states
- (b) separate MLP for each of the three blocks
- (c) skip connection



Recursive Network



Challenge of Long-Term Dependencies

- Recurrent networks involve the composition of the same function multiple times, once per time step
- The function composition resembles matrix multiplication: $h^{(t)} = W^T h^{(t-1)} = \cdots = (W^t)^T h^{(0)}$
- If W is decomposed into $Q\Lambda Q^T$ by an eigendecomposition, then $h^{(t)} = Q\Lambda^t Q^T h^{(0)}$
- This means the eigenvalues with magnitude less than one to decay to 0 and eigenvalues with magnitude greater than one to explode

- This leads to vanishing or exploding gradient problem

Strategies for Long-term Dependencies

- Design a model that operates at multiple time scales, so that some parts of the model operate at fine-grained time scales and can handle small details, while other parts operate at coarse time scales and transfer information from the distant past to the present more efficiently
  - Skip connections across time
  - “Leaky units” that integrate signals with different time constants
  - Removal of some of the connections used to model fine-grained time scales
  - Gated RNNs
    - Long Short-Term Memory (LSTM)
    - Gated Recurrent Unit (GRU)

Skip Connections through Time

- One way to obtain coarse time scales is to add direct connections from variables in the distant past to variables in the present
  - The idea is similar to that of ResNet
- Gradients may vanish or explode exponentially with respect to the number t of time steps
- Introducing recurrent connections with a time-delay of d makes gradient diminish exponentially as a function of t/d rather than t
- Since there are both delayed and single step connections, gradients may still explode exponentially in t
- This allows the learning algorithm to capture longer dependencies although not all long-term dependencies may be represented well in this way

Leaky Units

- When we accumulate a running average $\mu^{(t)}$ of some value $v^{(t)}$ by applying the update $\mu^{(t)} \leftarrow \alpha\mu^{(t-1)} + (1-\alpha)v^{(t)}$, the α parameter is an example of a linear self-connection from $\mu^{(t-1)}$ to $\mu^{(t)}$
  - When α is near 1, the running average remembers information about the past for a long time
  - When α is near 0, information about the past is rapidly discarded

- Leaky units: hidden units with linear self-connections
  - This approach allows to control the degree of using past information by adjusting α

Removing Connections

- Removing length-one connections and replacing them with longer connections
  - This is different from skip connections that add edges; units receiving such new connections may learn to operate on a long time scale but may also choose to focus on their other short-term connections

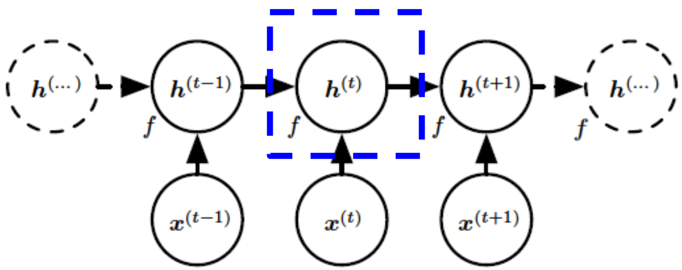
Gated RNNs

- Like leaky units, gated RNNS are based on the idea of creating paths through time that have derivatives that neither vanish nor explode
  - Leaky units did this with manually chosen connection weights; Gated RNNs allow the connection weights to change at each time step

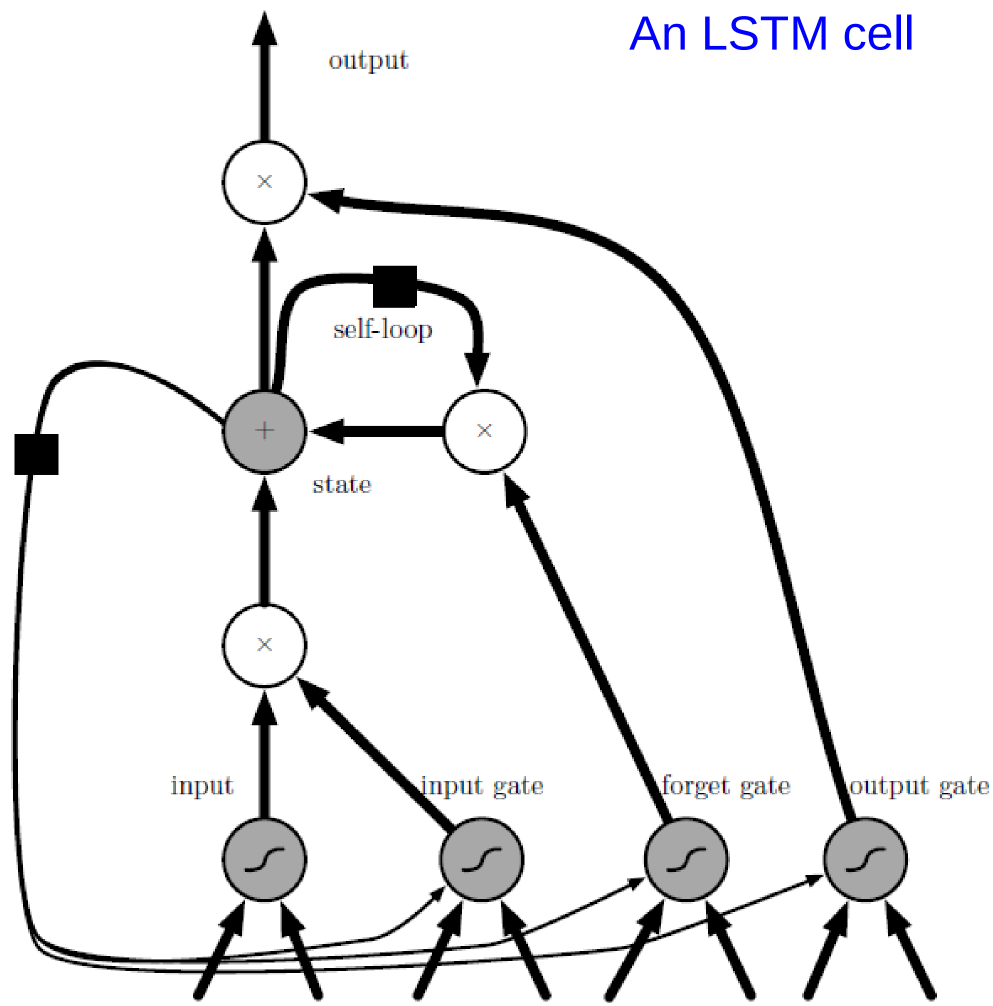
- Leaky units allow the network to accumulate information over time. However, once that information has been used, it might be useful to forget the old state
  - Gated RNNs learn to decide when to clear the old state

Long Short-Term Memory (LSTM)

- An LSTM recurrent network "cell" that replaces a hidden unit in a typical RNN

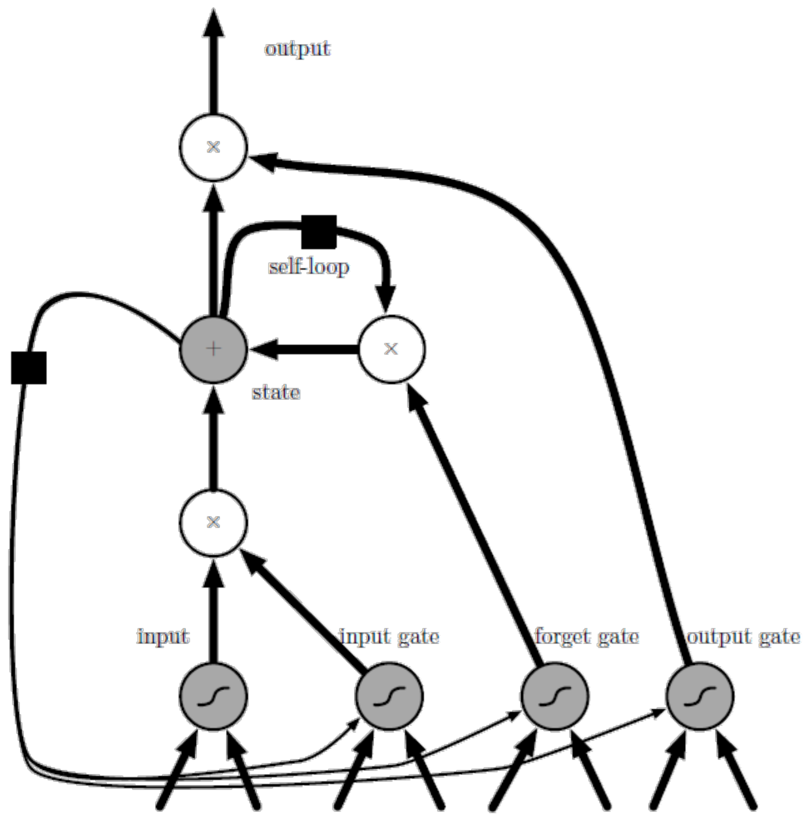


Each cell (e.g. $h^{(t)}$) in RNN receives input x and its previous state $h^{(t-1)}$ to make an output



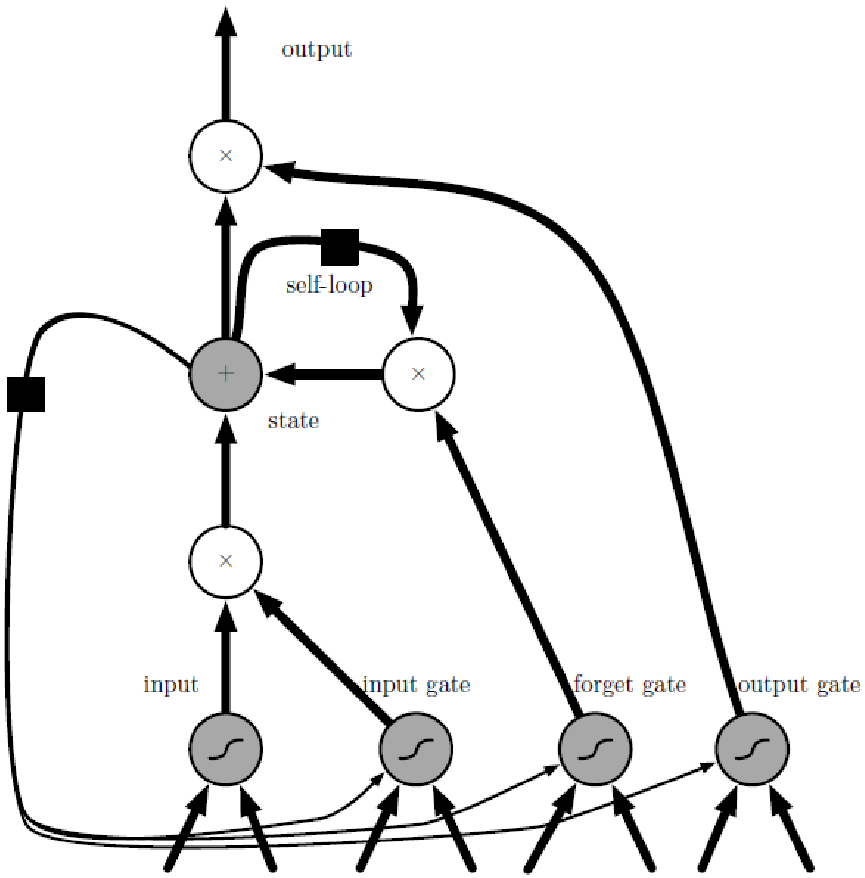
LSTM

- Initial LSTM (1997): introducing self-loops to produce paths where the gradient can flow for long durations
- (2000) Making the weight on this self-loop gated (controlled by another hidden unit)
- LSTM is a core module for many applications
  - Handwriting recognition
  - Speech recognition
  - Handwriting generation
  - Machine translation
  - Image captioning



LSTM

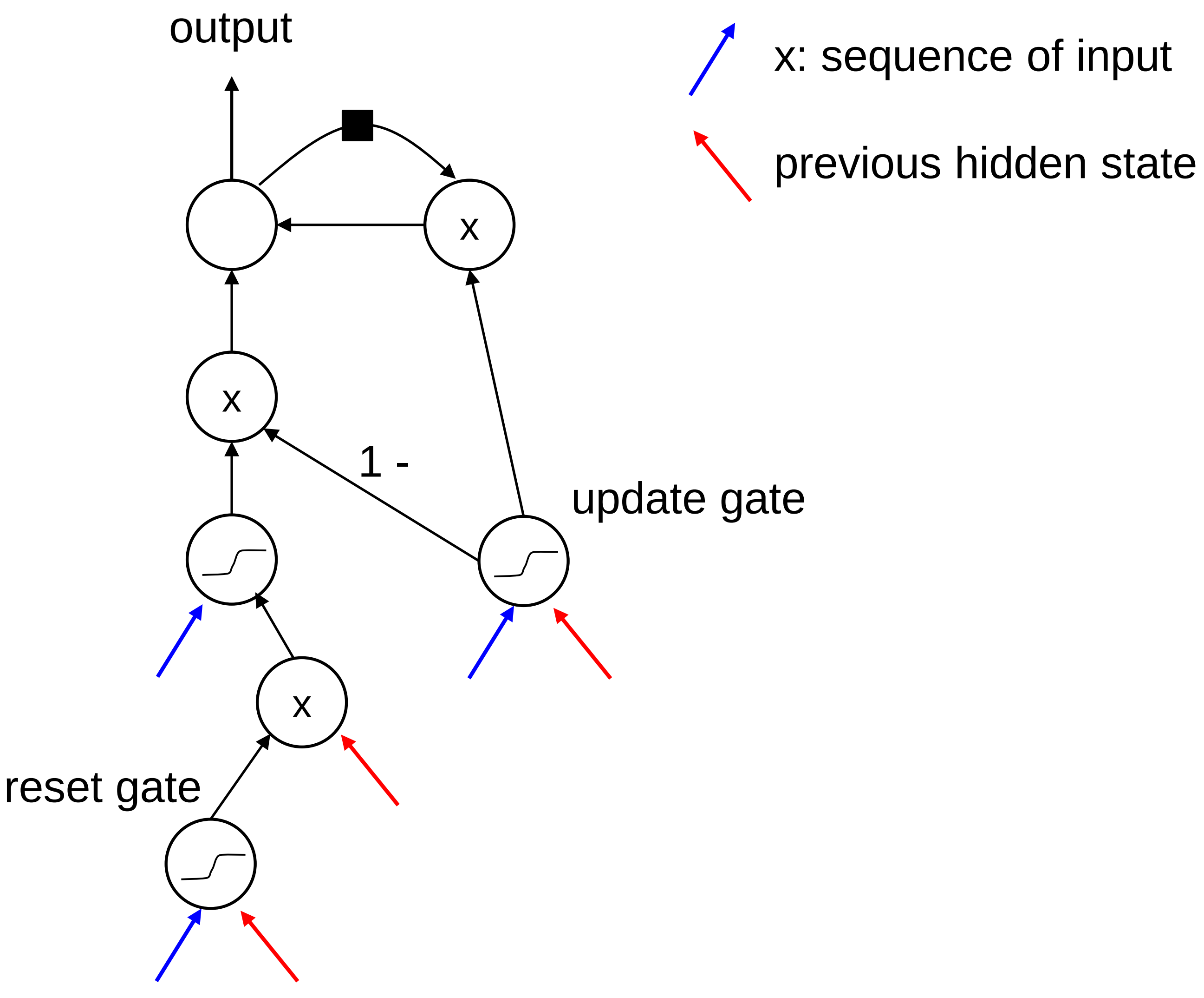
- Self-loop weight is controlled by a forget gate unit $f_i^{(t)}$ for time step t and cell i
  - $f_i^{(t)} = \sigma(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)})$
- The internal state $s_i^{(t)}$ is updated with a conditional self-loop weight $f_i^{(t)}$
  - $s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)})$
- The external input gate unit $g_i^{(t)}$ is computed similarly to the forget gate
  - $g_i^{(t)} = \sigma(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)})$
- The output $h_i^{(t)}$ of the LSTM cell can also be shut off, via the output gate $q_i^{(t)}$
  - $h_i^{(t)} = \tanh(s_i^{(t)}) q_i^{(t)}$
  - $q_i^{(t)} = \sigma(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)})$



Gated Recurrent Unit (GRU)

- Similar to LSTM; the main difference is that in GRU a single gating unit simultaneously controls the forgetting factor and the decision to update the state unit
  - $h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)}) \sigma(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)})$
- u stands for "update" gate and r for "reset" gate
  - $u_i^{(t)} = \sigma(b_i^u + \sum_j U_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)})$
  - $r_i^{(t)} = \sigma(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)})$
- GRU is less complex (computationally efficient) than LSTM while providing similar accuracy
  - GRU uses 2 gates, while LSTM uses 3 gates

GRU

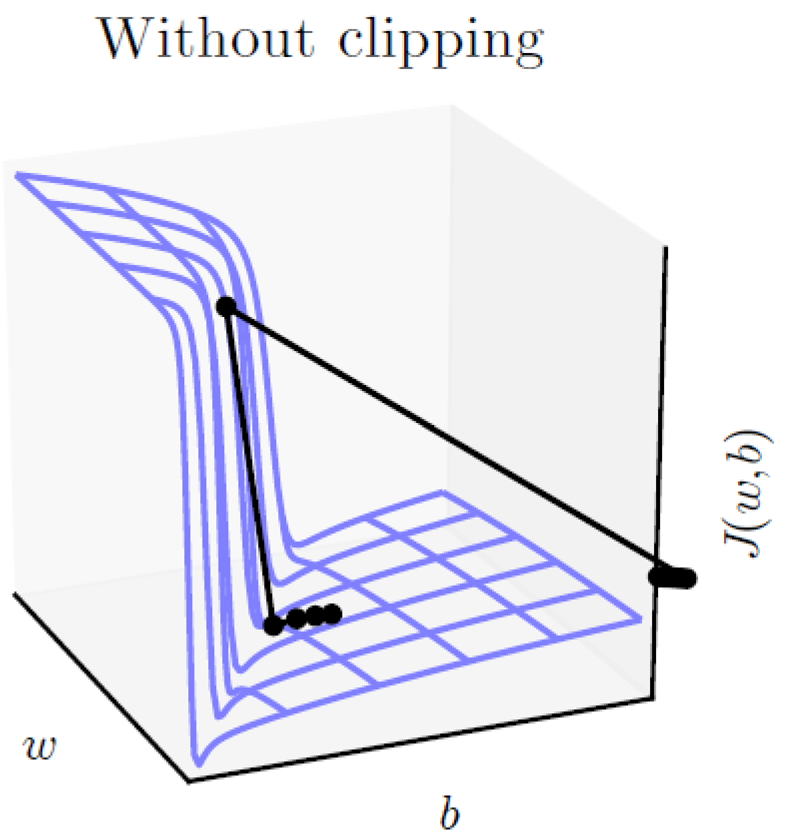

output
x: sequence of input
previous hidden state
x
x
1 -
update gate
x
reset gate

Optimization for Long-Term Dependencies

- Gradients of parameters in RNN can be very large due to long-term dependencies
- When the parameter gradient is very large, a gradient descent parameter update could throw the parameters very far, into a region where the objective function is larger, undoing much of the work that hand been done to reach the current solution
- Gradient clipping: a simple solution that avoids very large gradient
  - 2 versions
    - Clip the gradient element wise, just before the parameter update
    - Clip the norm $||g||$ of the gradient g, just before the parameter update
      - If $||g|| > v$, then $g \leftarrow \frac{gv}{||g||}$

v: norm threshold

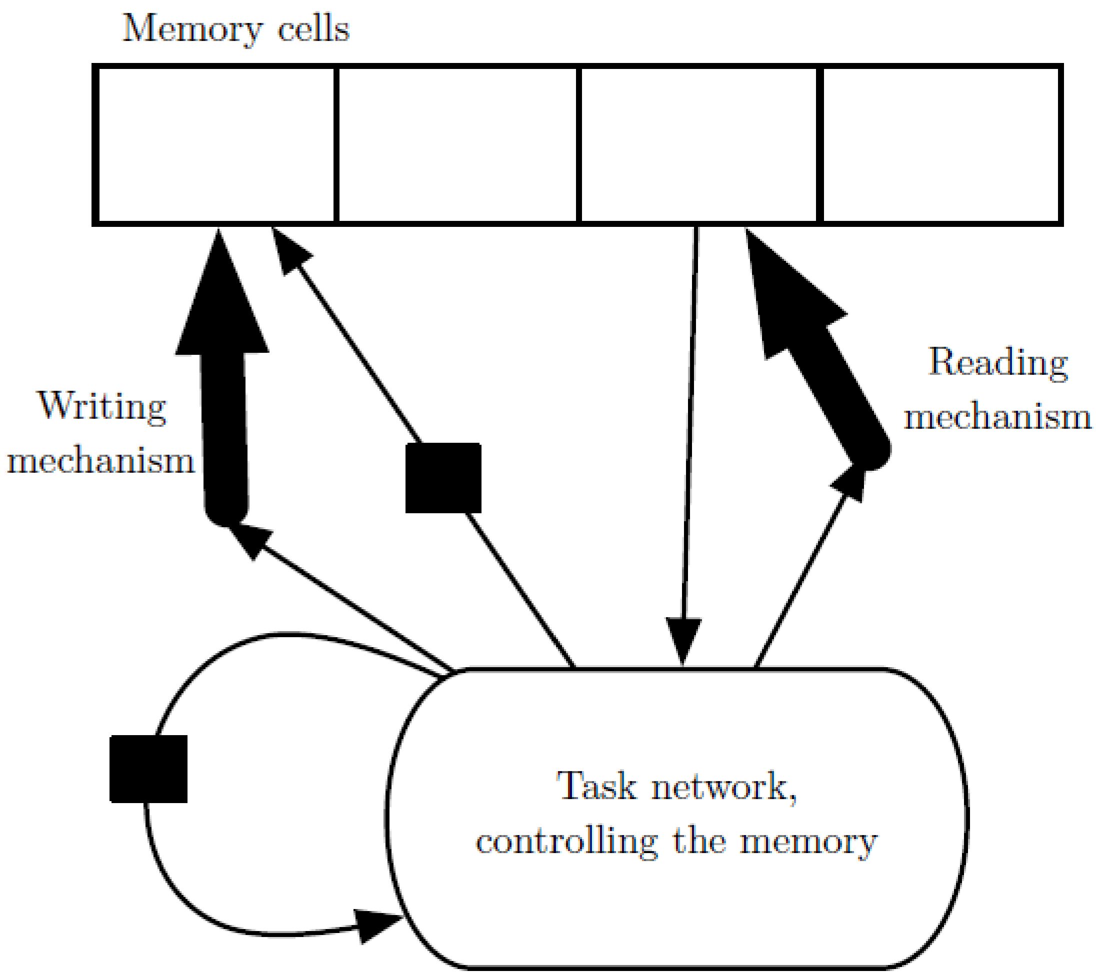
Gradient Clipping



Networks with Explicit Memory

- Different types of knowledge
  - Implicit: sub-conscious, and difficult to verbalize: e.g., how to walk, how a dog looks different from a cat
  - Explicit: declarative, and relatively straightforward to put into words. E.g., a cat is a kind of animal
- Neural networks excel at storing implicit knowledge. However, they struggle to memorize facts
  - The reason is because neural networks lack the working memory
- Memory networks: include a set of memory cells that can be accessed via an addressing mechanism
- Neural Turing machine: learns to read from and write arbitrary content to memory cells without explicit supervision about which actions to undertake, and allowed end-to-end training without this supervision signal

Networks with Explicit Memory



What you need to know

- Recurrent Neural Network
  - Main idea: parameter sharing over time
  - Major architectures:
  - Problem of long-term dependencies: vanishing or exploding gradient
    - Model that operates at a multiple time scale: LSTM
    - Optimization: gradient clipping

Questions?