Deep Learning

Recurrent Neural Networks

Lecturer: Duc Dung Nguyen, PhD.
Contact: nddung@hcmut.edu.vn

Faculty of Computer Science and Engineering
Hochiminh city University of Technology

Contents

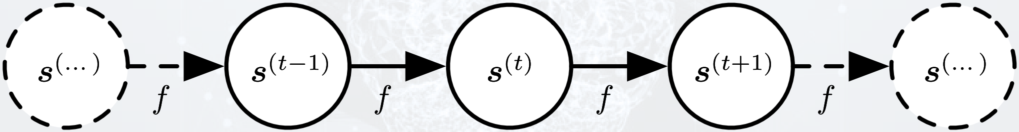
Unfolding Computational Graphs

- **Computational graph**: a way to formalize the structure of a set of computations
- Unfolding a recursive or recurrent computation into a computational graph that has a repetitive structure

- The classical form of a dynamical system

$$s^{(t)} = f(s^{(t-1)}; \theta) \tag{1}$$

where $s^{(t)}$ is called the state of the system

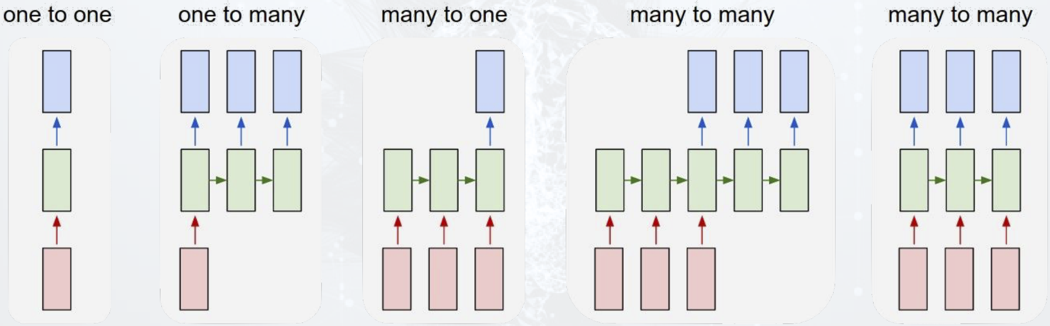

$s^{(\dots)}$
f
$s^{(t-1)}$
f
$s^{(t)}$
f
$s^{(t+1)}$
f
$s^{(\dots)}$

Unfolding Computational Graphs

- Any function involving recurrent can be considered as a feedforward network
- A single, shared model allows generalization to sequence lengths that did not appear in the training set
- Allows the model to be estimated with far fewer training examples
- The unfolded graph provides an explicit description of which computations to perform

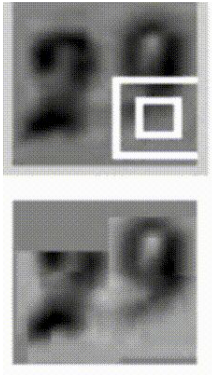
Recurrent Neural Networks

Motivation

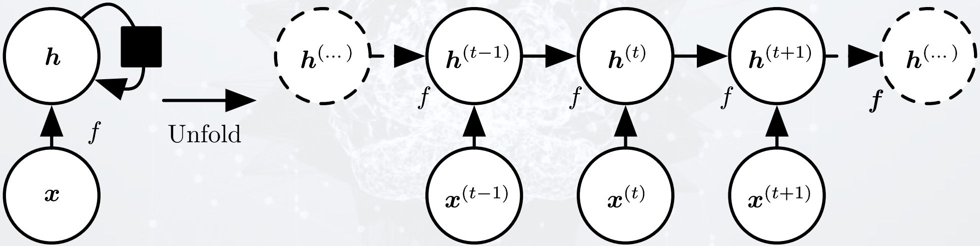


Vanilla Neural Networks

Motivation



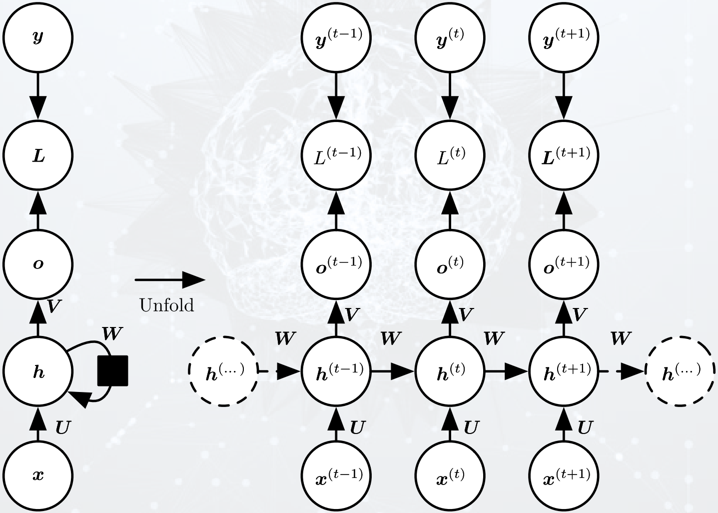
Recurrent Neural Networks



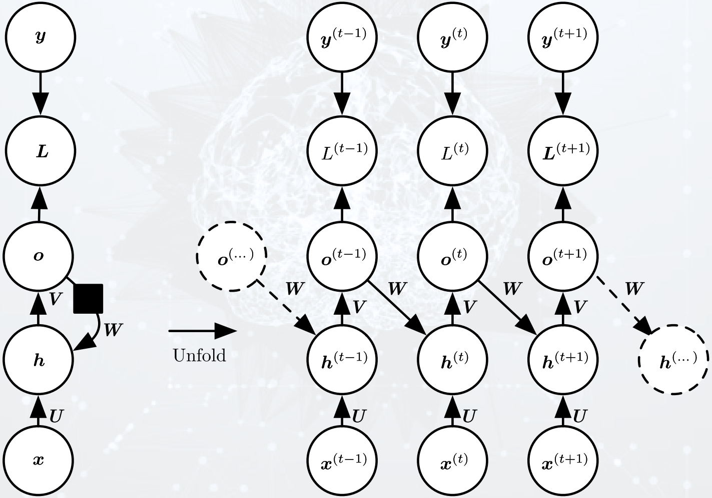
Some examples of important design patterns:

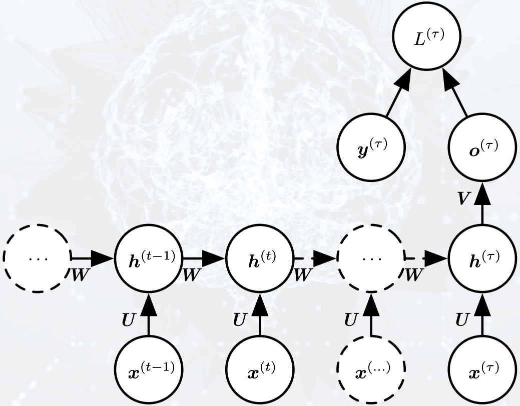
- RNNs that produce an output at each time step and have recurrent connections between hidden units
- RNNs that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step
- RNNs with recurrent connections between hidden units, that read an entire sequence and then produce a single output

Recurrent Neural Networks



Recurrent Neural Networks




$L^{(\tau)}$
$\boldsymbol{y}^{(\tau)}$
$\boldsymbol{o}^{(\tau)}$
$\boldsymbol{V}$
$\boldsymbol{h}^{(t-1)}$
$\boldsymbol{h}^{(t)}$
$\boldsymbol{h}^{(\tau)}$
$\boldsymbol{W}$
$\boldsymbol{U}$
$\boldsymbol{x}^{(t-1)}$
$\boldsymbol{x}^{(t)}$
$\boldsymbol{x}^{(\dots)}$
$\boldsymbol{x}^{(\tau)}$

Teacher Forcing and Networks with Output Recurrence

- Less powerful: lacks hidden-to-hidden recurrent connections
- Cannot simulate a universal Turing machine
- The output units are explicitly trained to matched the training set targets $\rightarrow$ unlikely to capture the necessary information about the past history
- Describe full state of the system?

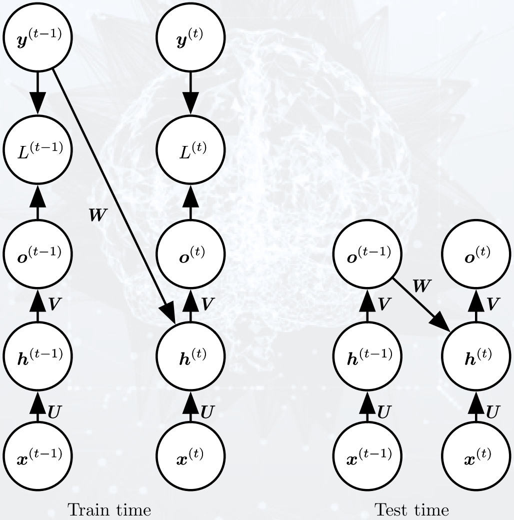
Teacher forcing

- A procedure that emerges from the **maximum likelihood criterion**: during training, the model receives the ground truth output $y(t)$ as input at time $t+1$.
- The conditional maximum likelihood criterion is

$$\log p\left(y^{(1)}, y^{(2)} | x^{(1)}, x^{(2)}\right) = \log p\left(y^{(2)} | y^{(1)}, x^{(1)}, x^{(2)}\right) + \log p\left(y^{(1)} | x^{(1)}, x^{(2)}\right) \tag{2}$$

- During training: feeding the model's own output back into itself $\rightarrow$ these connections should be fed with the target values specifying what the correct output should be

Teacher Forcing and Networks with Output Recurrence



Disadvantages

- When the network is going to be used in an open-loop mode $\rightarrow$ problem!
- Inputs in training can be quite different from the test time

Computing the Gradient in a Recurrent Neural Network

- **Training**: applies the generalized back-propagation algorithm to the unrolled computational graph
- **Back-propagation through time (BPTT)** algorithm

- The nodes of our computational graph include the parameters U, V, W, b and c
- The sequence of nodes are indexed by t for $x^{(t)}$, $h^{(t)}$, $o^{(t)}$ and $L^{(t)}$
- For each node N we need to compute the gradient $\nabla_N L$ recursively
- Start the recursion with the nodes immediately preceding the final loss

$$\frac{\partial L}{\partial L^{(t)}} = 1 \tag{3}$$

Computing the Gradient in a Recurrent Neural Network

- Assume:
  - Outputs $o^{(t)}$ are used as the argument to the softmax function to obtain the vector $\hat{y}$ of probabilities over the output
  - The loss is the negative log-likelihood of the true target $y^{(t)}$
- The gradient $\nabla_{o^{(t)}} L$ on the outputs at time step t, for all i,t:

$$(\nabla_{o^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - 1_{i,y^{(t)}} \tag{4}$$

Computing the Gradient in a Recurrent Neural Network

Back-propagation

- Starting from the end of the sequence
- At the final time step τ , $h^{(\tau)}$ only has $o^{(\tau)}$ as a descendent, so its gradient is simple:

$$\begin{aligned}\nabla_{h^{(t)}} L &= \left(\frac{\partial h^{(t+1)}}{\partial h^{(t)}}\right)^{\top} (\nabla_{h^{(t+1)}} L) + \left(\frac{\partial o^{(t)}}{\partial h^{(t)}}\right)^{\top} (\nabla_{o^{(t)}} L) \\ &= W^{\top} (\nabla_{h^{(t+1)}} L) \text{diag} \left(1 - \left(h^{(t+1)}\right)^{2}\right) + V^{\top} (\nabla_{o^{(t)}} L)\end{aligned} \tag{5}$$

Computing the Gradient in a Recurrent Neural Network

- How about $\nabla_W f$?
- Introduce dummy variables $W^{(t)}$ that are defined to be copies of W but with each $W^(t)$ used only at time step t
- Use $\nabla_{W^{(t)}}$ to denote the contribution of the weights at time step t to the gradient

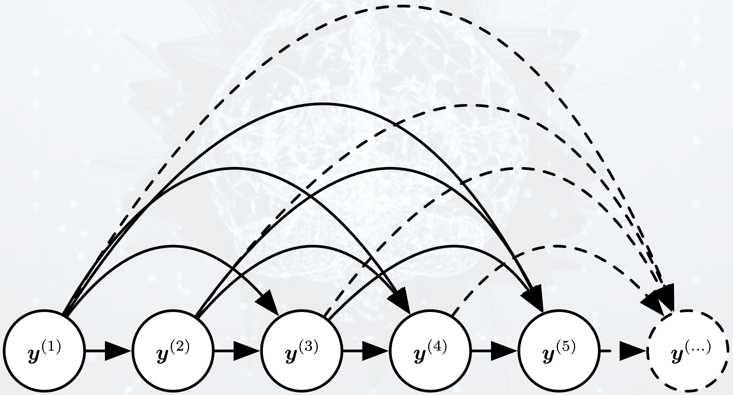
The gradient on the remaining parameters

$$
\begin{aligned}
\nabla_c L &= \sum_t \left(\frac{\partial o^{(t)}}{\partial c}\right)^\top \nabla_{o^{(t)}} L = \sum_t \nabla_{o^{(t)}} L \\
\nabla_b L &= \sum_t \left(\frac{\partial h^{(t)}}{\partial b^{(t)}}\right)^\top \nabla_{h^{(t)}} L = \sum_t \text{diag}\left(1 - \left(h^{(t)}\right)^2\right) \nabla_{h^{(t)}} L \\
\nabla_V L &= \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_V o_i^{(t)} = \sum_t (\nabla_{o^{(t)}} L) {h^{(t)}}^\top
\end{aligned}
\tag{6}
$$

Computing the Gradient in a Recurrent Neural Network

$$
\begin{aligned}
\nabla_W L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{W^{(t)}} h^{(t)} \\
&= \sum_t \text{diag}\left(1 - \left(h^{(t)}\right)^2\right) (\nabla_{h^{(t)}} L) h^{(t-1)^\top} \\
\nabla_U L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{U^{(t)}} h^{(t)} \\
&= \sum_t \text{diag}\left(1 - \left(h^{(t)}\right)^2\right) (\nabla_{h^{(t)}} L) x^{(t)^\top}
\end{aligned} \tag{7}
$$

Recurrent Networks as Directed Graphical Models



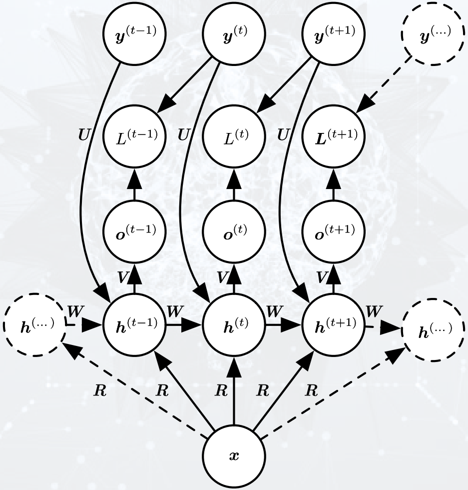
Recurrent Networks as Directed Graphical Models

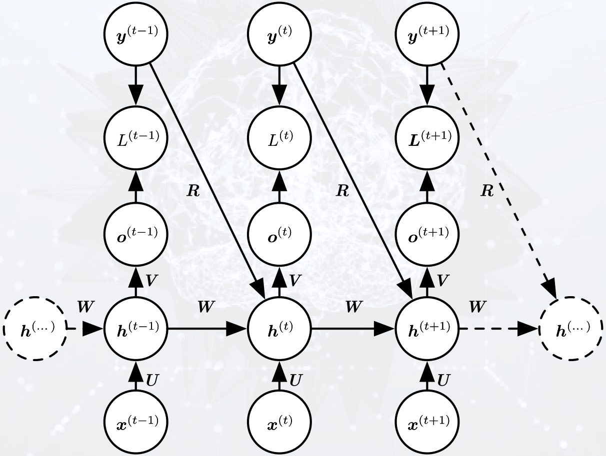
- Difficult to predict missing values in the middle of the sequence
- The price for the reduced number of parameters: optimizing the parameters may be difficult

Some common ways of providing an extra input to an RNN:

- As an extra input at each time step
- As the initial state $h^{(0)}$
- Both

Modeling Sequences Conditioned on Context with RNNs





$y^{(t-1)}$
$y^{(t)}$
$y^{(t+1)}$
$L^{(t-1)}$
$L^{(t)}$
$L^{(t+1)}$
R
R
R
$o^{(t-1)}$
$o^{(t)}$
$o^{(t+1)}$
V
V
V
W
W
W
W
$h^{(\dots)}$
$h^{(t-1)}$
$h^{(t)}$
$h^{(t+1)}$
$h^{(\dots)}$
U
U
U
$x^{(t-1)}$
$x^{(t)}$
$x^{(t+1)}$

Bidirectional RNNs

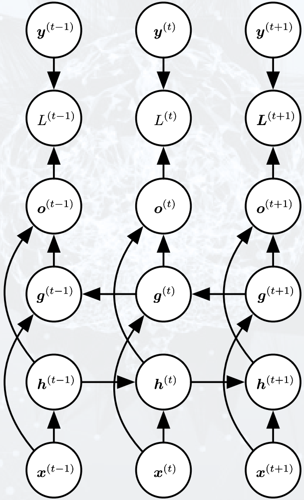
Bidirectional RNNs

- So far: "**causal**" structure
- What if a prediction of $y(t)$ depend on the whole input sequence?
- **Bidirectional RNNs**

Bidirectional RNNs

- Let remember the **past**
- And think about the **future**

Bidirectional RNNs



Encoder-Decoder Architectures

Encoder-Decoder Sequence-to-Sequence Architectures

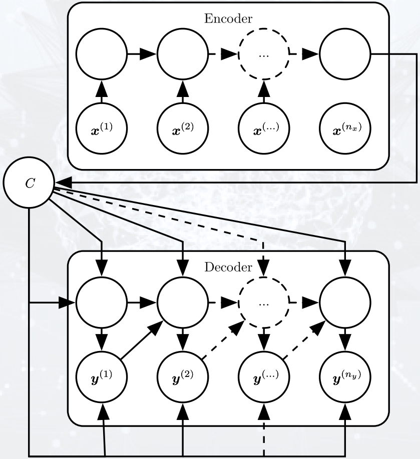
- Can we **map one sequence to another**?
- The input and the output are not necessary to have the same length
- **Examples?**

Examples

- Speech recognition
- Machine translation
- Question answering
- Etc.

- Input to RNN: **context**
- Produce a representation of context C
- **Context** C: a vector or sequence of vectors that summarize the input sequence $X = (x^{(1)}, ..., x^{(n_x)})$

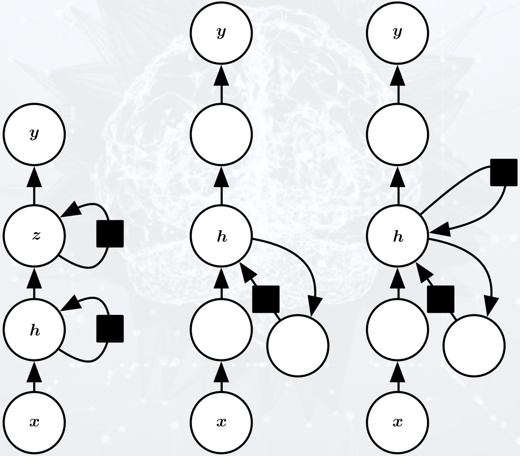
Encoder-Decoder Sequence-to-Sequence Architectures



- Encoder and decoder do not need to have the same size
- **Limitation**: the output of encoder has a dimension that is too small to properly summarize a long sequence

Other RNNs

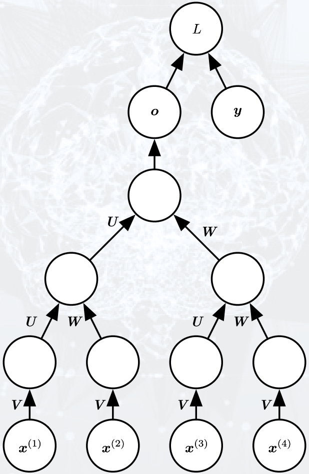
Deep Recurrent Networks



Recursive Neural Networks

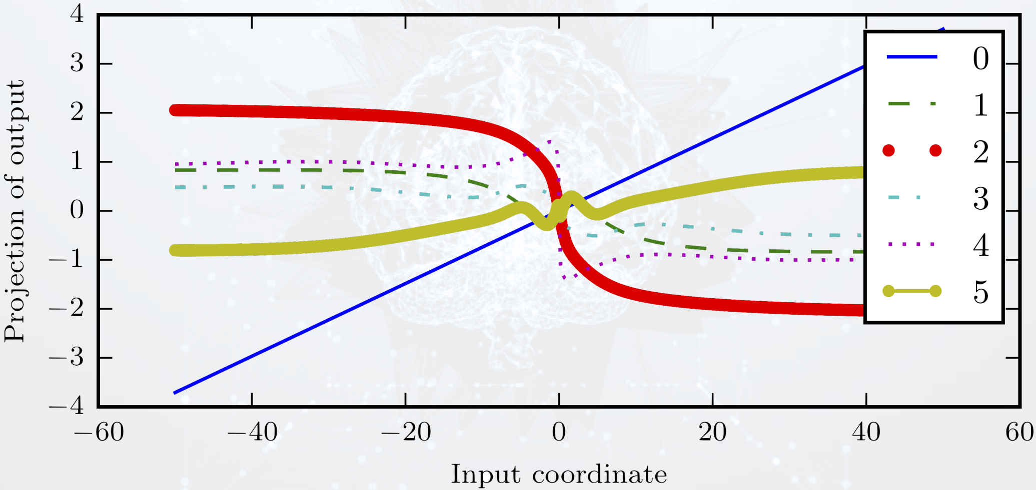
- A generalization of recurrent networks
- A different kind of computational graph: a deep tree, rather than the chain-like structure of RNNs
- E.g.: processing data structures as input to neural nets (both in NLP as well as in CV)

Recursive Neural Networks



- Advantage: for a sequence of the same length τ, the depth can be drastically reduced from τ to $O(\log \tau)$
- Might help deal with long-term dependencies
- **How to best structure the tree?**

The Challenge of Long-term Dependencies



The Long Short-Term Memory

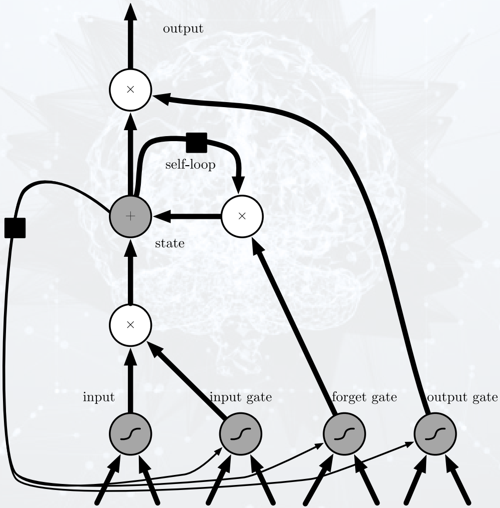
LSTM (Hochreiter and Schmidhuber, 1997)

- **LSTM**: introduce self-loops to produce paths where gradient can flow for a long durations.
- Make the **weight** on this self-loop **conditioned on the context**, rather than fixed
- **Gates**
- Time scale of integration can be changed dynamically

The Long Short-Term Memory

- The time scale of integration can change based on the input sequence
- The time constants are output by the model itself.

The Long Short-Term Memory



- Forget gate

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right) \tag{8}$$

- Internal state

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right) \tag{9}$$

- External input gate

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right) \tag{10}$$

- Output gate $q_i^{(t)}$

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)} \tag{11}$$

$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right) \tag{12}$$

The Long Short-Term Memory

- LSTM networks learns long-term dependencies better
- Optimization
  - Clipping gradient
  - Regularizing: encourage information flow
- Case studies:
  - Memory networks (Westion et al., 2014)
  - Neural Turing machine (Graves et al., 2014)
  - Multiple object recognition with attention (Ba et al.)
  - Image captioning