Recurrent Neural Networks

Prof. Ziping Zhao

School of Information Science and Technology
ShanghaiTech University, Shanghai, China

CS182: Introduction to Machine Learning (Fall 2021)
http://cs182.sist.shanghaitech.edu.cn

Outline

Introduction

Recurrent Neural Networks

RNN Generalizations

Long Short-Term Memory

Outline

Introduction

Recurrent Neural Networks

RNN Generalizations

Long Short-Term Memory

Introduction

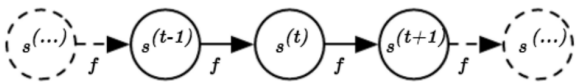
- Recurrent neural networks (RNNs) are extensions of feedforward neural networks for handling sequential data (such as sound, time series (sensor) data, or written natural language) typically involving variable-length input or output sequences.
- There is weight sharing in an RNN such that the weights are shared across different instances of the units corresponding to different time steps.
- Weight sharing is important for processing variable-length sequences so that the absolute time step at which an event occurs does not matter but the context (relative to some other events) in which an event occurs is more important and relevant.

Computational Graphs I

- The computation involved in a dynamical system defined recursively can be unfolded to form a computational graph with a repetitive structure such as a chain of events.
- A classical dynamical system:

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

  where $\mathbf{s}^{(t)}$ denotes the state of the system at time t and $\boldsymbol{\theta}$ denotes the system parameters which are time-invariant (effectively achieving weight sharing).
- Corresponding computational graph:

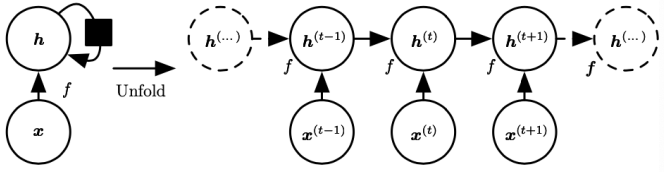


Computational Graphs II

- A more general dynamical system with an external input $\mathbf{x}^{(t)}$:

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \boldsymbol{\theta})$$

Unfolding a recurrent structure into a computational graph:



- Computations occurring at different time steps of the recurrent structure (where the black square indicates a delay of 1 time step) are represented as different nodes in the computational graph.
- $\mathbf{h}^{(t)}$ may be thought of as a kind of (lossy) summary of the past input sequences up to time t, i.e., $\mathbf{x}_\tau$ for $\tau \leq t - 1$.

Outline

Introduction

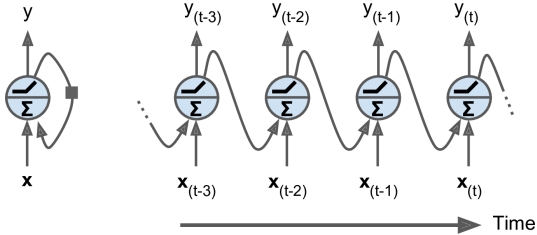
Recurrent Neural Networks

RNN Generalizations

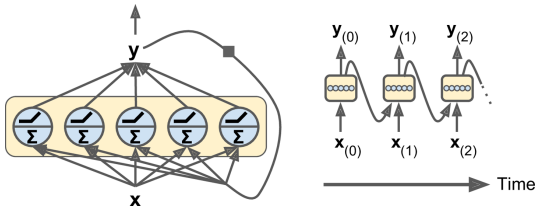
Long Short-Term Memory

Recurrent Neurons

- A recurrent neuron with a scalar output

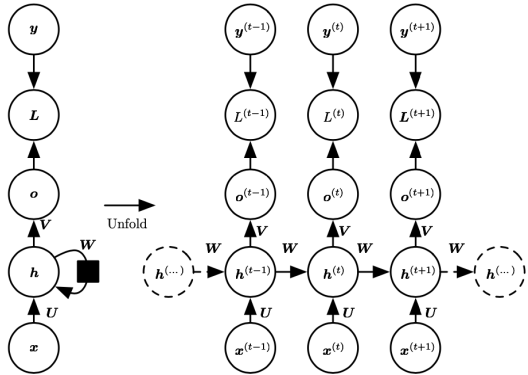


- A layer of recurrent neurons with a vector output



Computational Graphs for RNNs

- Computational graphs for showing the information flow in RNNs:
  - Forward in time: for computing outputs and losses
  - Backward in time: for computing gradients
- Unfolding an RNN (with hidden-to-hidden recurrence) into a computational graph:



Forward Propagation

- Forward propagation equations for classification problems:

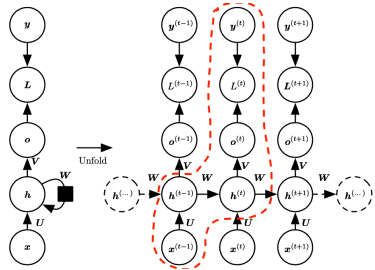
$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{U}\mathbf{x}^{(t-1)} + \mathbf{W}\mathbf{h}^{(t-1)}$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$

$$- - - - - - - - - - - - - -$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)}$$

$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{o}^{(t)})$$



where the weight matrices **U**, **V**, and **W** are the input-to-hidden, hidden-to-output, and hidden-to-hidden connections and **b** and **c** denote the bias vectors.

- In this model, the input sequence and the output sequence are of the same length.
- Total loss:

$$L(\mathbf{x}, \mathbf{y}) = \sum_t L^{(t)}$$

Back-Propagation Through Time

- The backpropagation learning algorithm for feedforward neural networks can be generalized to work on the computational graphs for RNNs, leading to the back-propagation through time (BPTT) algorithm.
- Different gradient-based techniques may be used by BPTT for RNN training.
- The following gradients (used by gradient-based techniques) can be computed recursively (details ignored) like the original backpropagation algorithm:

$$\frac{\partial L}{\partial \mathbf{U}}, \quad \frac{\partial L}{\partial \mathbf{V}}, \quad \frac{\partial L}{\partial \mathbf{W}}, \quad \frac{\partial L}{\partial \mathbf{b}}, \quad \frac{\partial L}{\partial \mathbf{c}}$$

Outline

Introduction

Recurrent Neural Networks

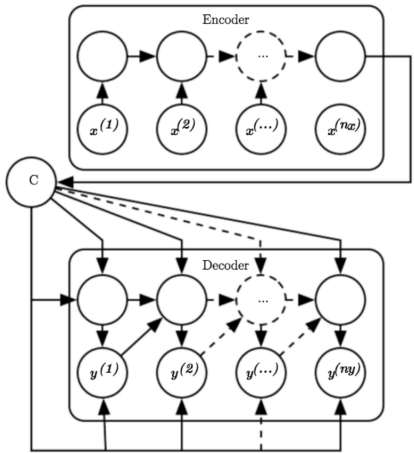
RNN Generalizations

Long Short-Term Memory

Encoder-Decoder RNN I

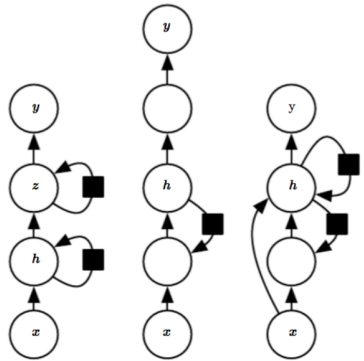
- An encoder-decoder RNN (or sequence-to-sequence learning architecture) learns to map an input sequence to an output sequence not necessarily of the same length (unlike the RNN above).
- Encoder RNN:
  - Processes the input sequence $\mathbf{X} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n_x)})$
  - Generates a (usually fixed-length) internal representation which is a simple function of the hidden state of the last hidden layer.
- Decoder RNN:
  - Takes the internal representation as input
  - Generates the output sequence $\mathbf{Y} = (\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(n_y)})$.
- In general the lengths n_x and n_y can vary from training pair to training pair.
- The two RNNs are trained jointly over all the training pairs.

Encoder-Decoder RNN II



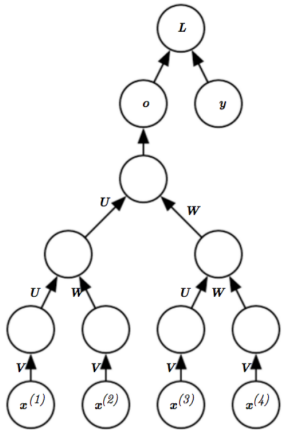
Deep RNNs

- Many ways to increase the network depth, e.g.,
  - Multiple recurrent hidden layers
  - Deeper computation in recurrent layers
  - Mixture of different recurrent connection types (path-lengthening effect may be mitigated by introducing skip connections)



Recursive Neural Networks I

- Recursive neural networks generalize ordinary RNNs in that the unfolded computational graphs are no longer chains but trees.



Recursive Neural Networks II

- Recursive neural networks can take data structures that are more general than feature vectors as input.
- They have been used successfully for natural language processing and computer vision applications. (You can take a course on "Deep Learning for Natural Language Processing".)
- To process a sequence of length N, an RNN has a length of N but a recursive neural network reduces it to $O(\log N)$. This helps the latter to deal with long-term dependencies more effectively.

Outline

Introduction

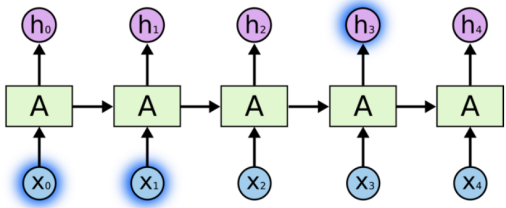
Recurrent Neural Networks

RNN Generalizations

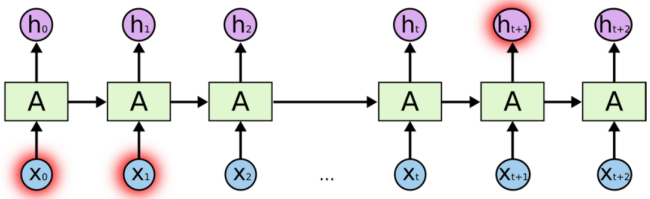
Long Short-Term Memory

Dependencies between Events in RNNs

- Short-term dependencies:



- Long-term dependencies:



Long-Term Dependencies

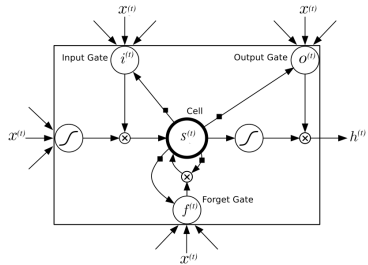
- Learning long-term dependencies in RNNs is challenging because:
  - the gradients propagated over many stages tend to vanish (most of the time) or explode (sometimes)
  - exponentially smaller weights are given to long-term interactions involving the multiplication of many Jacobians.
- The use of long short-term memory (LSTM) cells is one way proposed to help the learning of long-term dependencies in RNNs.

Long Short-Term Memory

- LSTM is based on the idea of leaky units which allow an RNN to accumulate information over a longer duration (e.g., we use a leaky unit to accumulate evidence for each subsequence inside a sequence).
- Once the information accumulated is used (e.g., suffcient evidence has been accumulated for a subsequence), we need a mechanism to forget the old state by setting it to zero and starting to count again from scratch.
- Instead of manually deciding when to clear the state, LSTM provides a way to learn to decide when to do it by conditioning the forgetting on the context.
- Gradients can flow for long durations through a linear self-loop whose weight is controlled by another hidden unit to change the time scale of the integration dynamically.

LSTM Cell

- An LSTM recurrent neural network consists of LSTM cells as hidden units which have an internal recurrence in addition to the recurrence in the recurrent network.
- 3 gates (or gate controllers) in an LSTM cell: external input gate, forget gate, output gate



a peephole LSTM cell

- LSTM RNN is a kind of gated RNN.
- Each LSTM cell has the same inputs and outputs as an ordinary RNN cell, but has more parameters and a system of gating units to control the information flow.

Forget and Input Gates

- For time step t, there is a state unit $s^{(t)}$ ("long-tem memory") that has a linear self-loop whose weight is controlled by a forget gate unit $f^{(t)}$ by setting the weight to a value in $(0, 1)$ via a sigmoid unit σ:

$$f^{(t)} = \sigma(b^f + U^f x^{(t)} + W^f s^{(t-1)})$$

  where $x^{(t)}$ is the current input vector, and b^f, U^f, and W^f are the biases, input weights, and recurrent weights of the forget gate.

- The external input gate unit $h^{(t)}$ is computed similarly to the forget gate:

$$i^{(t)} = \sigma(b^i + U^i x^{(t)} + W^i s^{(t-1)})$$

  where b^i, U^i, and W^i are the biases, input weights, and recurrent weights of the input gate.

Output Gate

- State update or forward equations:

$$s^{(t)} = f^{(t)} s^{(t-1)} + i^{(t)} \underbrace{\tanh(b + Ux^{(t)})}_{\tilde{s}^{(t)}}$$

  where **b** and **U** are the biases and input weights into the LSTM cell.

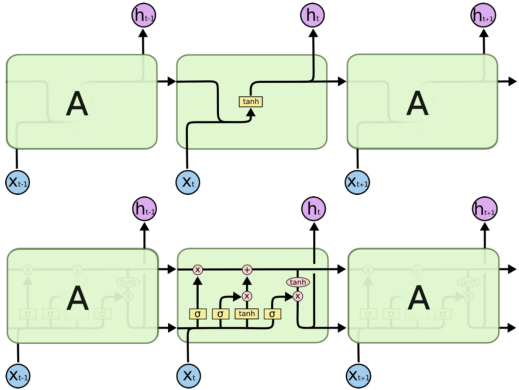
- The output gate unit $o^{(t)}$ can be used to shut off the output $h^{(t+1)}$ ("short-tem memory") of the LSTM cell:

$$o^{(t)} = \sigma(b^o + U^o x^{(t)} + W^o s^{(t-1)})$$
$$h^{(t)} = o^{(t)} \tanh(s^{(t)})$$

  where b^o, U^o, and W^o are its biases, input weights, and recurrent weight.

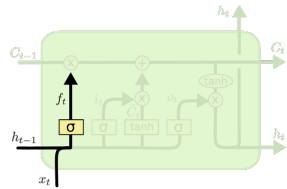
RNNs with ordinary cells vs. RNNs with LSTM cells

- Another structure of LSTM cell is as follows.



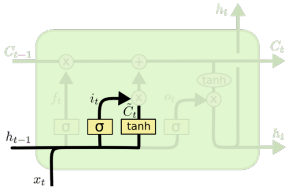
Schematic Illustration: Cell State and Forget Gate

- Forget gate:



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

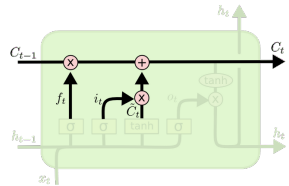
- External Input gate (we use C_t here for cell state instead of s_t used before): :



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

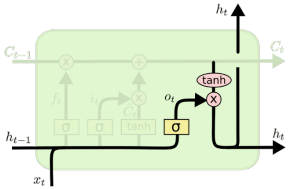
Schematic Illustration: Input Gate and Output Gate

- Cell state (we use C_t here for cell state instead of s_t used before):



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

- Output gate:



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