WestlakeNLP

Natural Language Processing

Yue Zhang
Westlake University

西湖大學
WESTLAKE UNIVERSITY

WestlakeNLP

Chapter 14

Representation Learning

Contents

WestlakeNLP

Contents

WestlakeNLP

Recurrent Neural Network

WestlakeNLP

Disadvantages of pooling and CNN

- Pooling only
  - Limited representation power
  - Insensitive to the input order
  - Cannot capture non-linear interactions between input vectors
- CNN
  - Cannot capture long-range dependencies between input vectors

Recurrent Neural Network

WestlakeNLP

Recurrent Neural Network (RNN)

- A recurrent state-transition process for left-to-right of the input sentence
- The state represents the syntactic, semantic and discourse context from the beginning until the current input
- Using a standard perceptron layer with non-linear activation to achieve the recurrent state-input combination function

Contents

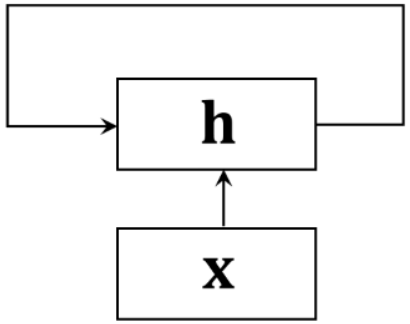
WestlakeNLP

Vanilla RNNs

WestlakeNLP

- An input sequence: $\mathbf{X}_{1:n} = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, n is the length of the sequence
- An initial state: $\mathbf{h}_0$ (set to **zero** or a randomly initialized model parameter)

- How to calculate an output sequence $\mathbf{h}_t (t \in [1, \ldots, n])$ using a vanilla RNN?

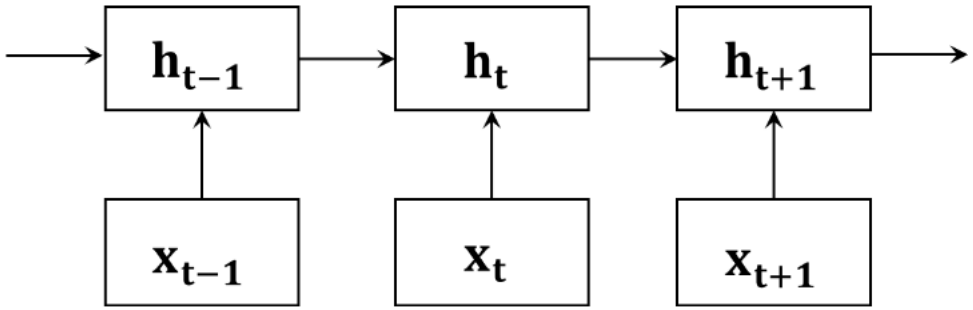


Vanilla RNNs

WestlakeNLP

- Given the **previous state** $\mathbf{h}_{t-1}$ and **the current input** $\mathbf{x}_t$, the **current state** $\mathbf{h}_t$ can be calculated as

$$\begin{aligned}\mathbf{h}_t &= \text{RNN\_STEP}(\mathbf{x}_t, \mathbf{h}_{t-1}) \\ &= f(\mathbf{W}^h \mathbf{h}_{t-1} + \mathbf{W}^x \mathbf{x}_t + \mathbf{b}),\end{aligned}$$



f: a **non-linear activation function** such as *tanh*

$\mathbf{W}^h$, $\mathbf{W}^x$,$\mathbf{b}$: model parameters, shared among different time steps

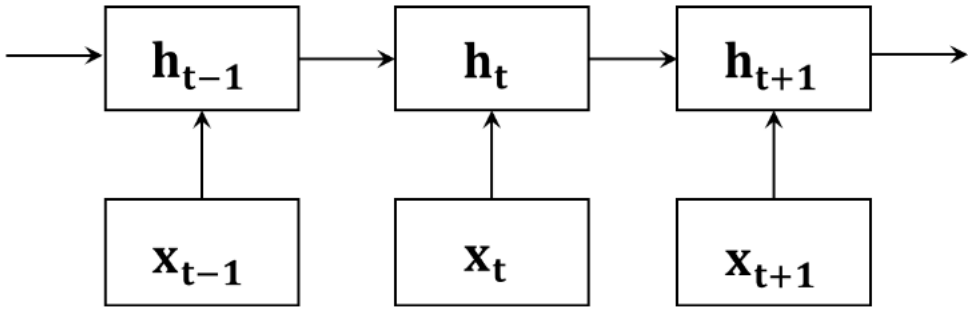
The **final vector** $\mathbf{h}_n$ can be used for representing the input $\mathbf{X}_{1:n}$.

Vanilla RNNs

WestlakeNLP

- Given the **previous state** $\mathbf{h}_{t-1}$ and **the current input** $\mathbf{x}_t$, the **current state** $\mathbf{h}_t$ can be calculated as

$$\begin{aligned}\mathbf{h}_t &= \text{RNN\_STEP}(\mathbf{x}_t, \mathbf{h}_{t-1}) \\ &= f(\mathbf{W}^h \mathbf{h}_{t-1} + \mathbf{W}^x \mathbf{x}_t + \mathbf{b}),\end{aligned}$$



f: a **non-linear activation function** such as *tanh*

$\mathbf{W}^h$, $\mathbf{W}^x$,$\mathbf{b}$: model parameters, shared among different time steps

The **final vector** $\mathbf{h}_n$ can be used for representing the input $\mathbf{X}_{1:n}$.

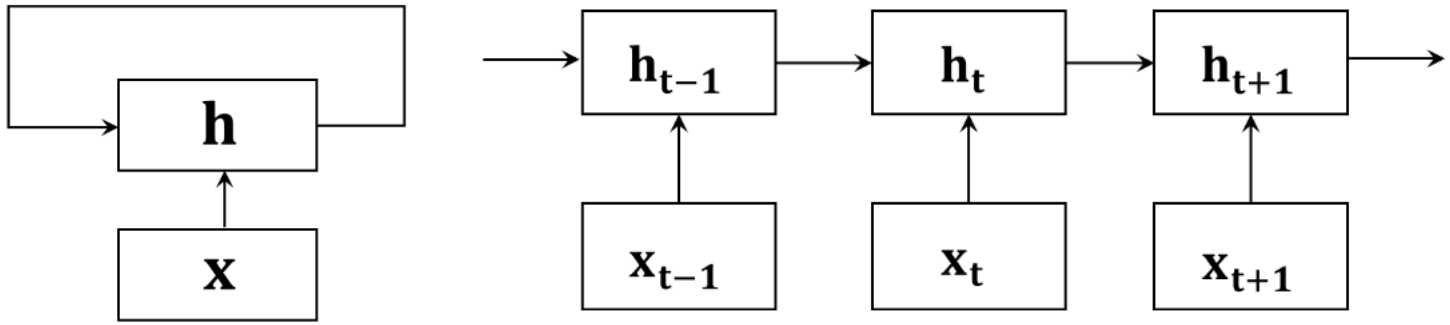
- We learned feed-forward processes.
- How do we understand a recurrent process?

Vanilla RNNs

WestlakeNLP

Layers and time steps

- A better understanding of RNNs: **exchanging time for space.**
- Viewed as "unfold": a standard multi-layer perceptron with lower layers towards the left and upper layers towards the right.
- The size of the network dynamically grows with the size of the input sequence.
- Sharing of model parameters across layers.



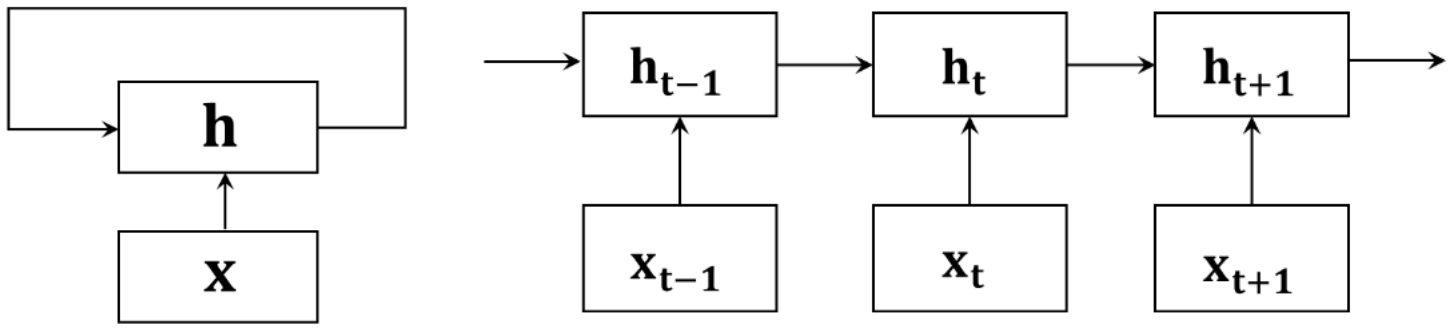
Original Unfold

Vanilla RNNs

WestlakeNLP

Layers and time steps

- Long-range dependency.
- Only contains the history on the left when encoding each word.



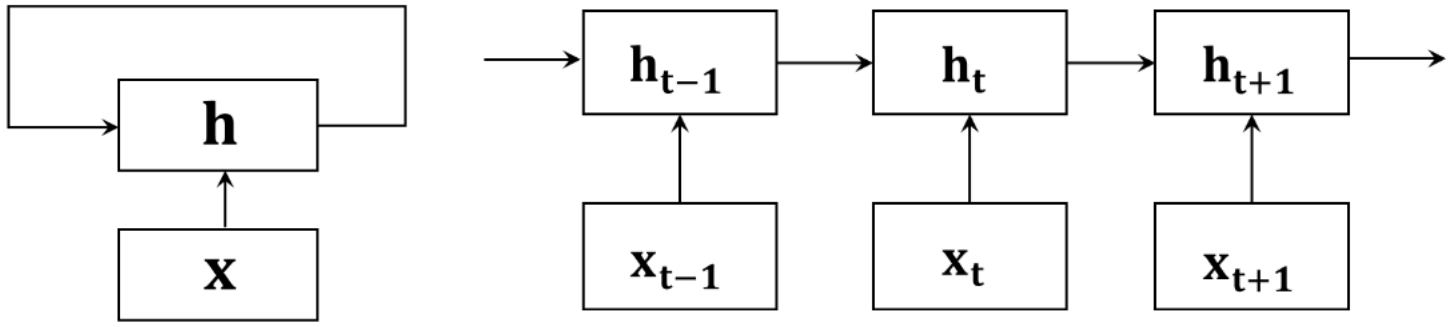
Original Unfold

Vanilla RNNs

WestlakeNLP

Output layer

- Use $\boldsymbol{h}_n$ as final $\boldsymbol{h}$
- Use pooling of $\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_n$



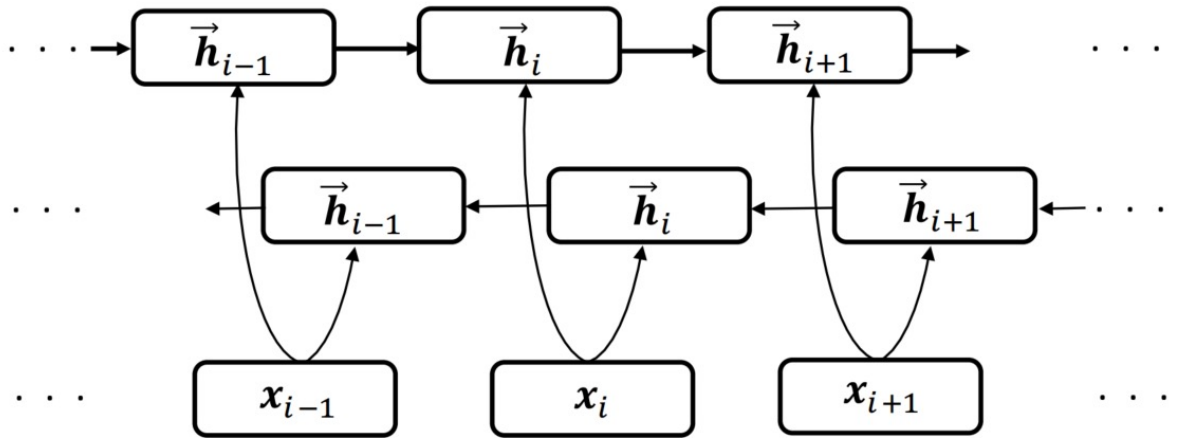
Original Unfold

Vanilla RNNs

WestlakeNLP

Bi-directional RNNs

- Concatenating the **historical context** using the left-to-right RNN ($\overrightarrow{RNN}$) and **model future information** using the right-to-left RNN ($\overleftarrow{RNN}$)
- Parameters of $\overrightarrow{RNN}$ and $\overleftarrow{RNN}$ can be different



An example of Bi-directional RNNs

Vanilla RNNs

WestlakeNLP

Bi-directional RNNs

Denote a bi-directional RNN by the function $BiRNN(\mathbf{X})$:

$$\overrightarrow{\mathbf{H}} = \overrightarrow{RNN}(\mathbf{X}) = [\vec{\mathbf{h}}_1; \vec{\mathbf{h}}_2; \ldots; \vec{\mathbf{h}}_n]$$

$$\overleftarrow{\mathbf{H}} = \overleftarrow{RNN}(\mathbf{X}) = [\overleftarrow{\mathbf{h}}_1; \overleftarrow{\mathbf{h}}_2; \ldots; \overleftarrow{\mathbf{h}}_n]$$

$$BiRNN(\mathbf{X}) = \overrightarrow{\mathbf{H}} \oplus \overleftarrow{\mathbf{H}} = [\vec{\mathbf{h}}_1 \oplus \overleftarrow{\mathbf{h}}_1; \vec{\mathbf{h}}_2 \oplus \overleftarrow{\mathbf{h}}_2; \ldots; \vec{\mathbf{h}}_n \oplus \overleftarrow{\mathbf{h}}_n]$$

- $\oplus$: the vector concatenation operation
- A concatenation of the left-to-right feature vector $\vec{\mathbf{h}}_t$ and the right-to-left feature vector $\overleftarrow{\mathbf{h}}_t$ gives the final representation of the t-th word representation

Vanilla RNNs

WestlakeNLP

Bi-directional RNNs

Output layer

- Use $\overrightarrow{\boldsymbol{h}_n} \oplus \overleftarrow{\boldsymbol{h}_n}$ as final $\boldsymbol{h}$.
- Use pooling of $\overrightarrow{\boldsymbol{h}_1} \oplus \overleftarrow{\boldsymbol{h}_1}$, $\overrightarrow{\boldsymbol{h}_2} \oplus \overleftarrow{\boldsymbol{h}_2}$ …, $\overrightarrow{\boldsymbol{h}_n} \oplus \overleftarrow{\boldsymbol{h}_n}$.

Contents

WestlakeNLP

Training RNNs

WestlakeNLP

- Supposing that $\overrightarrow{\boldsymbol{h}_n}$ is used as final hidden state for RNN.
- Loss pass to $\boldsymbol{h}_n$
- Need loss for $\boldsymbol{W}^h, \boldsymbol{W}^b, \boldsymbol{b}$ and also $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$

Training RNNs

WestlakeNLP

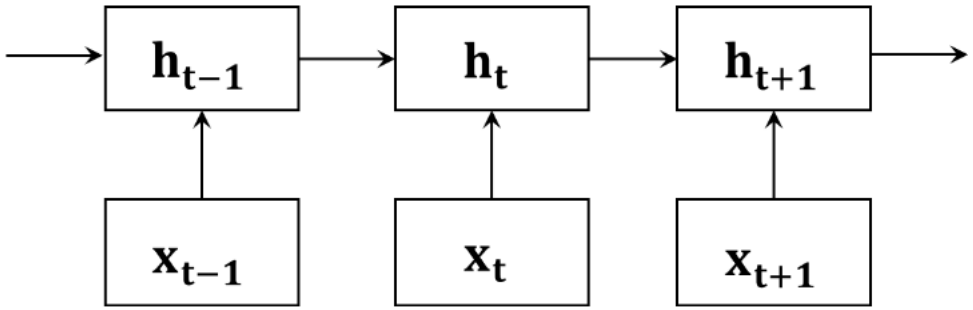
Back-propagation through time (BPTT)

- RNNs are trained using unfolded representation with **back-propagation through time** (BPTT).
- Assuming that the activation function is

$$f = \tanh$$

- The RNNs forward-propagation computing returns as

$$\mathbf{h}_t = \tanh(\mathbf{W}^h \mathbf{h}_{t-1} + \mathbf{W}^x \mathbf{x}_t + \mathbf{b})$$



Training RNNs

WestlakeNLP

Back-propagation through time (BPTT)

Given a vector value $\frac{\partial L}{\partial \mathbf{h}_t}$ passed down from layers above, BTTP returns results as follows:

$$\frac{\partial L}{\partial \mathbf{x}_t} = (\mathbf{W}^x)^T \cdot \left(\frac{\partial L}{\partial \mathbf{h}_t} \otimes (1 - \mathbf{h}_t^2)\right)$$

$$\frac{\partial L}{\partial \mathbf{h}_{t-1}} = (\mathbf{W}^h)^T \cdot \left(\frac{\partial L}{\partial \mathbf{h}_t} \otimes (1 - \mathbf{h}_t^2)\right)$$

$$\frac{\partial L}{\partial \mathbf{W}^h} = \left(\frac{\partial L}{\partial \mathbf{h}_t} \otimes (1 - \mathbf{h}_t^2)\right) \cdot \mathbf{h}_{t-1}^T$$

$$\frac{\partial L}{\partial \mathbf{W}^x} = \left(\frac{\partial L}{\partial \mathbf{h}_t} \otimes (1 - \mathbf{h}_t^2)\right) \cdot \mathbf{x}_t^T$$

$$\frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{h}_t} \otimes (1 - \mathbf{h}_t^2),$$

$\otimes$: element-wise product

Training RNNs

WestlakeNLP

Gradient issues

RNNS can be difficult to train using SGD due to **gradient exploding** and **gradient vanishing** problems.

For $\frac{\partial L}{\partial \mathbf{h}_{n-t}}$ with a relatively large number t, we have

$$
\begin{aligned}
\frac{\partial L}{\partial \mathbf{h}_{n-t}} &= (\mathbf{W}^h)^T \cdot \left(\frac{\partial L}{\partial \mathbf{h}_{n-t+1}} \otimes (1-\mathbf{h}_{n-t+1}^2) \right) \\
&= (\mathbf{W}^h)^T \cdot \left((\mathbf{W}^h)^T \cdot \left(\frac{\partial L}{\partial \mathbf{h}_{n-t+2}} \otimes (1-\mathbf{h}_{n-t+2}^2) \right) \otimes (1-\mathbf{h}_{n-t+1}^2) \right) \\
&= \quad \cdots \\
&= \left((\mathbf{W}^h)^T \right)^t \cdot \frac{\partial L}{\partial \mathbf{h}_n} \left(\otimes_{j=1}^{t} (1-\mathbf{h}_{n-t+j}^2) \right)
\end{aligned}
$$

Training RNNs

WestlakeNLP

Gradient issues

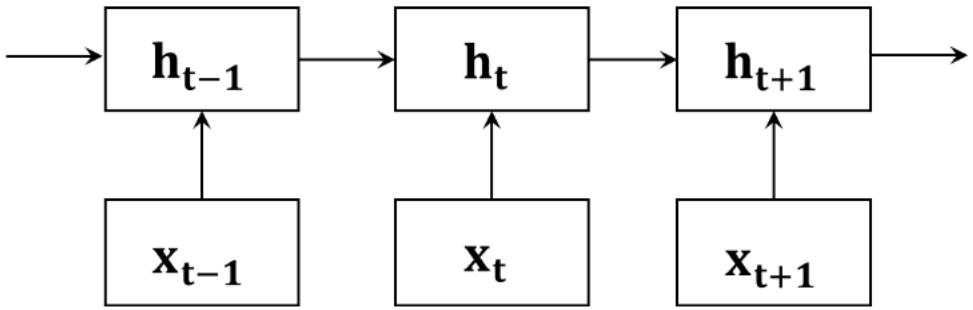
- Reasons for Vanishing gradients
  - Due to $1 - h_{n-t+j}^2 \in [0,1]$, $\otimes_{j=1}^{t} (1 - \mathbf{h}_{n-t+j}^2)$ can be extremely small;
  - $(\mathbf{W}^h)^T$ is not initialized properly with a **small value**, $\left((\mathbf{W}^h)^T\right)^t$ can be very **small**
- Reasons for exploding gradients
  - $(\mathbf{W}^h)^T$ is not initialized properly with a **large value**, $\left((\mathbf{W}^h)^T\right)^t$ can be very **large**

Training RNNs

WestlakeNLP

Tricks for avoiding gradient issues

- Using **truncated BPTT** to mitigate the gradient exploding problem
- Using appropriate weight initializations
- Using alternative RNN models such as GRUs and LSTMs



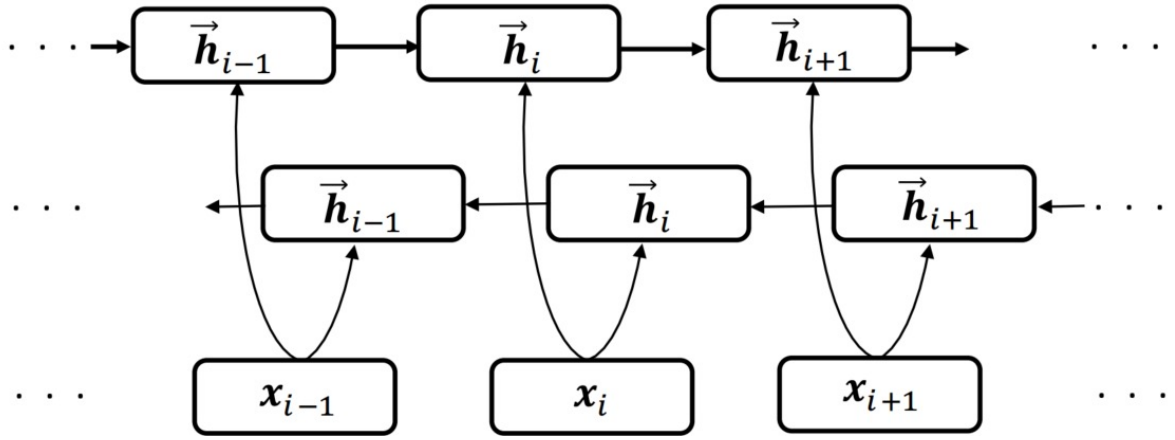
Training RNNs

WestlakeNLP

Training bi-directional RNNs

Two different aspects from training BiRNNs and vanilla RNNs:

- Both $\overrightarrow{\mathbf{h}_n}$ and $\overleftarrow{\mathbf{h}_1}$ receive back-propagated gradients
- Each $\mathbf{x}_i$ $(i \in [1, \ldots, n])$ receives back-propagated gradients from both $\overrightarrow{\mathbf{h}_i}$ and $\overleftarrow{\mathbf{h}_i}$. These two gradients should be summed as the final gradient.



An example of Bi-directional RNNs

Contents

WestlakeNLP

Long-Short-Term Memory

WestlakeNLP

Long-short-term memory (LSTM):

- An RNN variant which allows better SGD training by better control of back-propagation gradients over a large number of steps;
- Splitting the hidden state of each recurrent step into a *state vector* and a *memory cell vector*.
- Using gates for fine-grained control of "remembered" and "forgotten" information by each feature

Long-Short-Term Memory

WestlakeNLP

Given an input $\mathbf{X}_{1:n}$, the state vector $\mathbf{H}_{1:n}$ and cell vectors (representing a recurrent memory in LSTM) $\mathbf{C}_{1:n}$, with randomly initialized model parameters $\mathbf{h_0}$ (initial state) and $\boldsymbol{c_0}$(cell vectors),

How to calculate the standard LSTM step

$$\boldsymbol{h}_t, \boldsymbol{c}_t = \text{LSTM\_STEP}(\mathbf{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1}) \text{ ?}$$

Long-Short-Term Memory

WestlakeNLP

A standard LSTM recurrent step can be calculated as follows:

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}^{ih}\mathbf{h}_{t-1} + \mathbf{W}^{ix}\mathbf{x}_t + \mathbf{b}^i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}^{fh}\mathbf{h}_{t-1} + \mathbf{W}^{fx}\mathbf{x}_t + \mathbf{b}^f) \\
\mathbf{g}_t &= \tanh(\mathbf{W}^{gh}\mathbf{h}_{t-1} + \mathbf{W}^{gx}\mathbf{x}_t + \mathbf{b}^g) \\
\mathbf{c}_t &= \mathbf{i}_t \otimes \mathbf{g}_t + \mathbf{f}_t \otimes \mathbf{c}_{t-1} \\
\mathbf{o}_t &= \sigma(\mathbf{W}^{oh}\mathbf{h}_{t-1} + \mathbf{W}^{ox}\mathbf{x}_t + \mathbf{b}^o) \\
\mathbf{h}_t &= \mathbf{o}_t \otimes \tanh(\mathbf{c}_t)
\end{aligned}
$$

$\mathbf{W}^{ih}, \mathbf{W}^{ix}, \mathbf{b}^i, \mathbf{W}^{fh}, \mathbf{W}^{fx}, \mathbf{b}^f, \mathbf{W}^{gh}, \mathbf{W}^{gx}, \mathbf{b}^g, \mathbf{W}^{oh}, \mathbf{W}^{ox}$ and $\mathbf{b}^o$ are model parameters;

$\mathbf{g}_t$: nonlinear transformation for better representing the input $\mathbf{x}_t$;

$\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t$: input gate, forget gate and output gate, respectively;

σ , $\otimes$: the sigmoid function and the element-wise multiplication (i.e., Hadamard product) operation, respectively.

Long-Short-Term Memory

WestlakeNLP

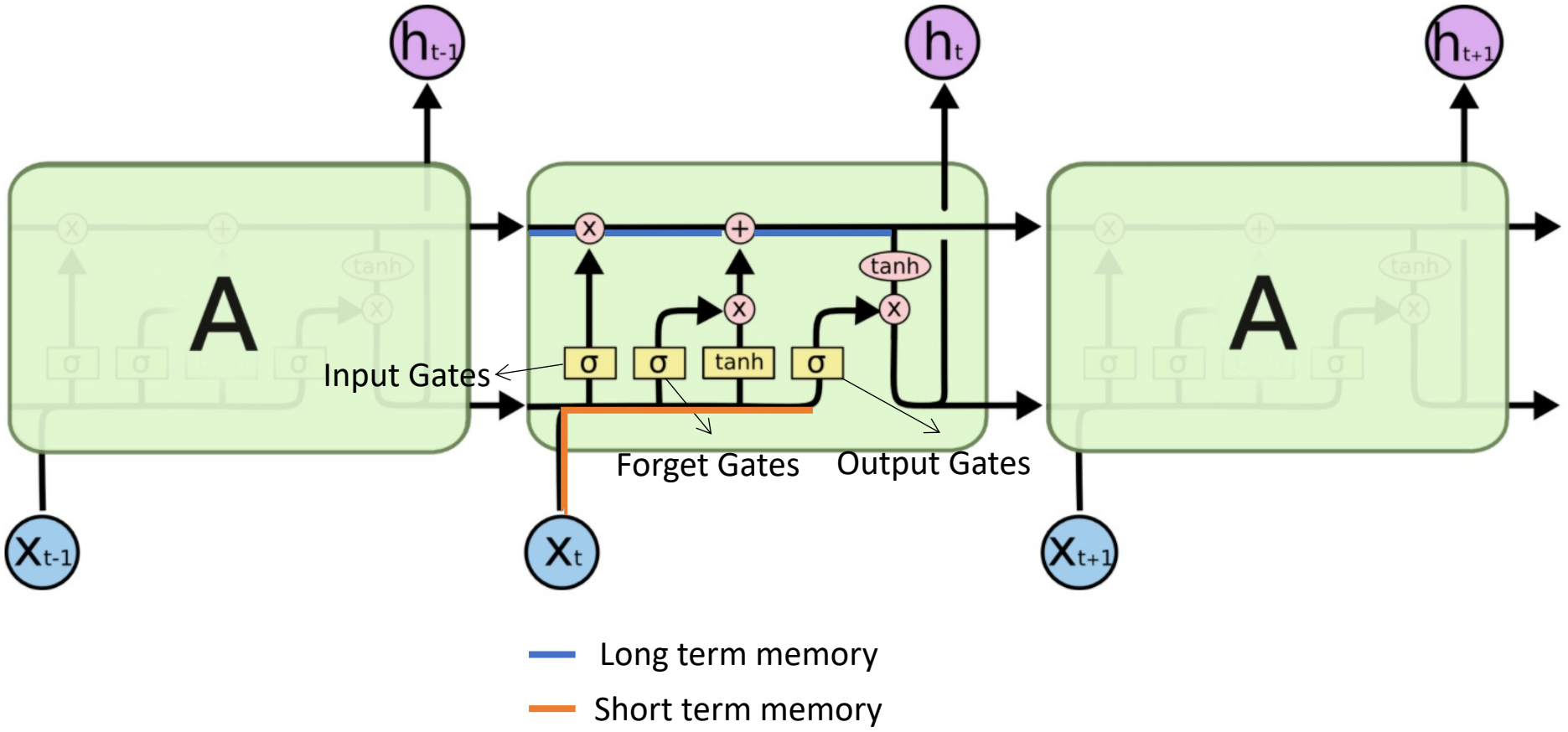


Figure credit: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long-Short-Term Memory

WestlakeNLP

Gates in LSTM

- LSTM recurrent steps are characterized by the use of gates through the **Hadamard product operation**
- A **gate vector** takes a real value between **0 and 1**
- The element-wise product of a gate vector and a feature vector filters each feature with a decay

Long-Short-Term Memory

WestlakeNLP

Gates in LSTM

- Input gate ($\mathbf{i}_t$): controls the reading process of the current input
- Forget gate ($\mathbf{f}_t$): keeps the history in memory
- Output gate ($\mathbf{o}_t$): decides the mapping from a memory cell to a hidden vector

Long-Short-Term Memory

WestlakeNLP

Bi-directional extension

The **bi-directional LSTMs** (BiLSTM) can be defined as follows:

$$\overrightarrow{\mathbf{H}} = \overrightarrow{LSTM}(\mathbf{X}) = [\overrightarrow{\mathbf{h}}_1; \overrightarrow{\mathbf{h}}_2; \ldots; \overrightarrow{\mathbf{h}}_n],$$

$$\overleftarrow{\mathbf{H}} = \overleftarrow{LSTM}(\mathbf{X}) = [\overleftarrow{\mathbf{h}}_1; \overleftarrow{\mathbf{h}}_2; \ldots; \overleftarrow{\mathbf{h}}_n],$$

$$BiLSTM(\mathbf{X}) = \overrightarrow{\mathbf{H}} \oplus \overleftarrow{\mathbf{H}} = [\overrightarrow{\mathbf{h}}_1 \oplus \overleftarrow{\mathbf{h}}_1; \overrightarrow{\mathbf{h}}_2 \oplus \overleftarrow{\mathbf{h}}_2; \ldots; \overrightarrow{\mathbf{h}}_n \oplus \overleftarrow{\mathbf{h}}_n],$$

$\overrightarrow{LSTM}$: left-to-right LSTMs

$\overleftarrow{LSTM}$: right-to-left LSTMs

Gated recurrent units

WestlakeNLP

Gated recurrent Units (GRU)

- Compared to RNNs, LSTMs give better results, but much slower due to increased model parameters and computation steps;
- **Gated recurrent units (GRU)** simplify LSTM by removing the cell structure, and using only two gates (a reset gate and a forget gate)
- Better deal with back-propagation gradients with a faster speed

Gated recurrent units

WestlakeNLP

Given an input sequence: $\mathbf{X}_1^n = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, a standard GRU cell $\mathbf{h}_t = \text{GRU\_STEP}(\mathbf{x}_t, \mathbf{h}_{t-1})$ is given by

$$\mathbf{r}_t = \sigma(\mathbf{W}^{rh}\mathbf{h}_{t-1} + \mathbf{W}^{rx}\mathbf{x}_t + \mathbf{b}^r)$$
$$\mathbf{z}_t = \sigma(\mathbf{W}^{zh}\mathbf{h}_{t-1} + \mathbf{W}^{zx}\mathbf{x}_t + \mathbf{b}^z)$$
$$\mathbf{g}_t = \tanh(\mathbf{W}^{hh}(\mathbf{r}_t \otimes \mathbf{h}_{t-1}) + \mathbf{W}^{hx}\mathbf{x}_t + \mathbf{b}^h)$$
$$\mathbf{h}_t = (\mathbf{1.0} - \mathbf{z}_t) \otimes \mathbf{h}_{t-1} + \mathbf{z}_t \otimes \mathbf{g}_t,$$

$\mathbf{W}^{rh}$, $\mathbf{W}^{rx}$, $\mathbf{b}^r$, $\mathbf{W}^{zh}$, $\mathbf{W}^{zx}$, $\mathbf{b}^z$, $\mathbf{W}^{hh}$, $\mathbf{W}^{hx}$ and $\mathbf{b}^h$: model parameters
$\mathbf{r}_t$: the reset gate
$\mathbf{z}_t$: the forget gate

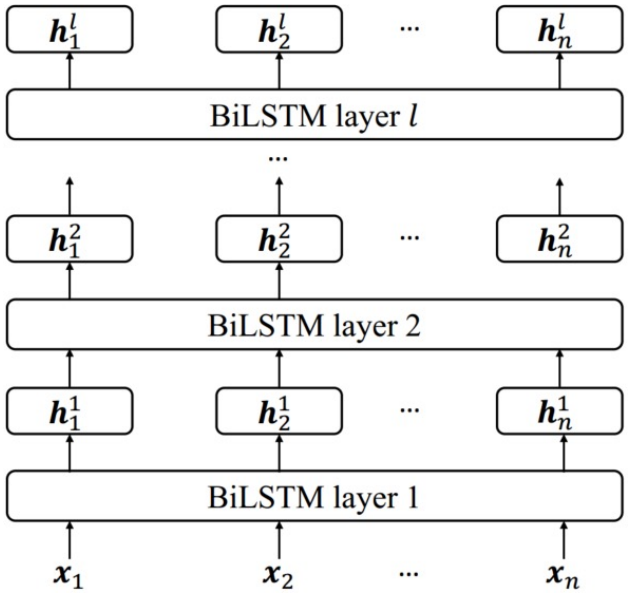
Contents

WestlakeNLP

Stacked LSTMs

WestlakeNLP

- Recurrent neural networks can be stacked to multiple layers to improve the representation power
- Each layer in stacked LSTMS feeds its output vectors as input to the next layer in the bottom-up direction.



An Example of Stacked BiLSTMs

Stacked LSTMs

WestlakeNLP

A stacking method can be calculated as follows:

$$\overrightarrow{\mathbf{H}}^0 = \mathbf{X}, \overleftarrow{\mathbf{H}}^0 = \mathbf{X}$$

$$\overrightarrow{\mathbf{H}}^1 = \overrightarrow{LSTM}_1(\overrightarrow{\mathbf{H}}^0), \overrightarrow{\mathbf{H}}^2 = \overrightarrow{LSTM}_2(\overrightarrow{\mathbf{H}}^1), \ldots, \overrightarrow{\mathbf{H}}^l = \overrightarrow{LSTM}_l(\overrightarrow{\mathbf{H}}^{l-1})$$

$$\overleftarrow{\mathbf{H}}^1 = \overleftarrow{LSTM}_1(\overleftarrow{\mathbf{H}}^0), \overleftarrow{\mathbf{H}}^2 = \overleftarrow{LSTM}_2(\overleftarrow{\mathbf{H}}^1), \ldots, \overleftarrow{\mathbf{H}}^l = \overleftarrow{LSTM}_l(\overleftarrow{\mathbf{H}}^{l-1})$$

$$\mathbf{H} = \overrightarrow{\mathbf{H}^l} \oplus \overleftarrow{\mathbf{H}^l} = [\overrightarrow{\mathbf{h}}_1^l \oplus \overleftarrow{\mathbf{h}}_1^l; \overrightarrow{\mathbf{h}}_2^l \oplus \overleftarrow{\mathbf{h}}_2^l; \ldots; \overrightarrow{\mathbf{h}}_n^l \oplus \overleftarrow{\mathbf{h}}_n^l]$$

$\mathbf{h}_t^j$: the output hidden vector of the t-th word at the j-th layer

$\mathbf{H}^j$: the output hidden vectors of the whole sequence at the j-th layer

$\mathbf{H}$: the final output vectors

$\overrightarrow{LSTM}_j$ and $\overleftarrow{LSTM}_j$: left-to-right LSTM and the right-to-left LSTM at the j-th layer, respectively

Contents

WestlakeNLP

Neural Attention

WestlakeNLP

Neural Attention

- An alternative method to pooling operations for aggregating a set of vectors
- A weighted sum of vectors in a sequence with regard to certain targets
- Can be used to find a single vector representation of a sentence

Neural Attention

WestlakeNLP

Given a target vector $\mathbf{q}(\mathbf{q} \in \mathbb{R}^d)$ and a list of context vectors $\mathbf{H} = \mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_n$ ($\mathbf{h}_i \in \mathbb{R}^d$, d is the dimension of $\mathbf{q}$), the function can be defined as:

$$s_i = score(\mathbf{q}, \mathbf{h}_i) \quad (i \in [1, ..., n])$$

$$\alpha_i = \frac{\exp(s_i)}{\sum_{i=1}^{N} \exp(s_i)} \quad \text{(softmax normalization)}$$

$$\mathbf{c} = \sum_{i=1}^{n} \alpha_i \times \mathbf{h}_i \quad \text{(weighted sum),}$$

$\mathbf{c}$: output of $attention(\mathbf{q}, \mathbf{H})$, a weighted sum of the content vectors, which can be used as a context-aware feature representation of $\mathbf{q}$

s_i: a relevance score between $\mathbf{q}$ and $\mathbf{h}_i$

α_i: normalised relevance scores based on s_i

$\alpha = [\alpha_1, \alpha_2, \cdots, \alpha_n]$: a probability distribution over the content vectors

Neural Attention

WestlakeNLP

Score function

Dot-product attention

- Defines the score between the target vector **q** and the context vector **h**
- No model parameters
- measures the similarity between **q** and **h**

$$score(\mathbf{q}, \mathbf{h}) = \mathbf{q}^T\mathbf{h}$$

Neural Attention

WestlakeNLP

Score function

Scaled dot-product attention

Scales the dot-product attention score by $\frac{1}{\sqrt{d}}$, where d is the dimension of $\mathbf{q}$ and $\mathbf{h}$

$$score(\mathbf{q}, \mathbf{h}) = \frac{\mathbf{q}^T \mathbf{h}}{\sqrt{d}}$$

Neural Attention

WestlakeNLP

Score function

General attention

A parameter matrix $\mathbf{W}$ ($\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$) to capture the interaction between each element in $\mathbf{q}$ ($\mathbf{q} \in \mathbb{R}^{d_1}$) and each element in $\mathbf{h}$ ($\mathbf{h} \in \mathbb{R}^{d_2}$)

$$score(\mathbf{q}, \mathbf{h}) = \mathbf{q}^T \mathbf{W} \mathbf{h}$$

Neural Attention

WestlakeNLP

Score function

Additive attention

- First performs a linear combination of **q** and **h**
- then applies a feedforward neural layer before squeezing the resulting vector using a parameter vector **v**

$$score(\mathbf{q}, \mathbf{h}) = \mathbf{v}^T \tanh(\mathbf{W}(\mathbf{q} \oplus \mathbf{h}) + \mathbf{b}),$$

v, **W**, **b** are model parameters

$\oplus$ denotes concatenation

Neural Attention

WestlakeNLP

Score function

- For *dot-production attention* and *scaled dot-production attention*, **q** and **h** must have the same dimension size;
- For *general attention* and *additive attention*, **q** and **h** can have different dimension size

Back-propagation Rules

WestlakeNLP

- Loss over $\boldsymbol{c}$ given
- Calculate loss over $\boldsymbol{q}, \boldsymbol{h}_i$ $(i \in [1, \ldots, n])$

Neural Attention

WestlakeNLP

Correlation with gating functions

- Given a set of hidden vectors $\mathbf{H}_{1:n} = \mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n$ and a target vector $\mathbf{q}$, a set of gate vectors for aggregating $\mathbf{H}_{1:n}$ can be calculated as

$$\mathbf{s}_i = \mathbf{W}^q \mathbf{q} + \mathbf{W}^h \mathbf{h}_i$$

$$\mathbf{g}_i = softmax(\mathbf{s}_1, \mathbf{s}_2, \ldots \mathbf{s}_n) \qquad \text{(element-wise softmax)}$$

$$\mathbf{c} = \sum_{i=1}^{n} \mathbf{g}_i \otimes \mathbf{h}_i,$$

$\mathbf{W}^q$ and $\mathbf{W}^h$ are model parameters

$\otimes$ denotes element-wise multiplication

- Offering more fine-grained combination of input vectors, but is also computationally more expensive

Contents

WestlakeNLP

Query-Key-Value Attention

WestlakeNLP

- Similar to database queries, contexts in neural attention also contain a set of key-value pairs;
- Context vectors can be regarded as associated memories in this case
- Given a **target query**, comparing the query vector with the key vectors and return the related value vectors

Query-Key-Value Attention

WestlakeNLP

- Suppose: the query vector $\mathbf{q}$, the key vector $\mathbf{K}_{1:n} = [\mathbf{k}_1; \mathbf{k}_2; \cdots; \mathbf{k}_n]$ and the value vector $\mathbf{V}_{1:n} = [\mathbf{v}_1; \mathbf{v}_2; \cdots; \mathbf{v}_n]$
- For each key vector $\mathbf{k}_i$, the corresponding value vector $\mathbf{v}_i$, the query-key-value attention function $attention(\mathbf{q}, \mathbf{K}, \mathbf{V})$ is

$$s_i = score(\mathbf{q}, \mathbf{k}_i) \quad (i \in [1, \ldots, n])$$

$$\alpha_i = \frac{\exp(s_i)}{\sum_{i=1}^{N} \exp(s_i)} \quad \text{(softmax normalization)}$$

$$\mathbf{c} = \sum_{i=1}^{n} \alpha_i \mathbf{v}_i \quad \text{(weighted sum),}$$

$\mathbf{c}$: output of $attention(\mathbf{q}, \mathbf{H})$, a weighted sum of the value vectors with the. i-th weight score being s_i, s_i: attention score between the query vector $\mathbf{q}$ and the i-th key vector $\mathbf{k}_i$

Query-Key-Value Attention

WestlakeNLP

Query-key-value attention with a sequence of queries

- Deal with *sequence* of queries: call call the attention function separately for each query, and then concatenate the results
- Given the sequence of queries $\mathbf{Q}_{1:l} = [\mathbf{q}_1; \mathbf{q}_2; \cdots; \mathbf{q}_l]$, key vectors K and value vectors V, the attention function $attention(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ is

$$\mathbf{c}_1 = attention(\mathbf{q}_1, \mathbf{K}, \mathbf{V})$$
$$\mathbf{c}_2 = attention(\mathbf{q}_2, \mathbf{K}, \mathbf{V})$$
$$\cdots$$
$$\mathbf{c}_l = attention(\mathbf{q}_l, \mathbf{K}, \mathbf{V})$$
$$attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\mathbf{c}_1; \mathbf{c}_2; \cdots; \mathbf{c}_l],$$

$\mathbf{c}_i \in \mathbb{R}^d$: the attentive result of the i-th query

Query-Key-Value Attention

WestlakeNLP

Parallel computations

Using matrix multiplications to enable parallel computations for reducing computational expenses

$$\mathbf{S} = score(\mathbf{Q}, \mathbf{K})$$

$$\mathbf{A} = softmax_1(\mathbf{S})$$

$\mathbf{C} = \mathbf{V}\mathbf{A}^T$: final result, which is taken as $attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \in \mathbb{R}^{l \times d}$. The i-th row represents the attentive result vector of $\mathbf{q}_i$

$\mathbf{S} \in \mathbb{R}^{l \times n}$: a score matrix, $s_{[i][j]}$(also donated as s_{ij}) is the relevance score of $\mathbf{q}_i$ and $\mathbf{k}_j$

$softmax_1(\mathbf{S})$: applying the softmax function to normalize each column in $\mathbf{S}$

$\mathbf{A} \in \mathbb{R}^{l \times n}$: attention score matrix

Contents

WestlakeNLP

Self-Attention-Network

WestlakeNLP

Self-Attention-Network (SAN)

- **Self-Attention-Network (SAN)** aggregates a set of vectors, which can be useful to design an attention network structure
- Given $\mathbf{X}_{1:n} = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, the output vector $\mathbf{H}_{1:n} = attention(\mathbf{X}_{1:n}, \mathbf{X}_{1:n}, \mathbf{X}_{1:n})$ can be calculated as

$$\mathbf{H}_{1:n} = attention(\mathbf{X}_{1:n}, \mathbf{X}_{1:n}, \mathbf{X}_{1:n})$$

$\mathbf{h}_i$: an attentive representation of $\mathbf{X}_{1:n}$ by **using $\boldsymbol{x_i}$ as a query**

Self-Attention-Network

WestlakeNLP

Two advantages for SANs

- Allowing the representation $\mathbf{h}_i$ in each layer to take into consideration all $\mathbf{x}_i$s globally
- The time complexity of RNNs is $O(n^n)$, while the time complexity of SANs is $O(n)$
- Transfer (Chapter 16) is a more advanced SAN framework.

Contents

WestlakeNLP

Representing Trees

WestlakeNLP

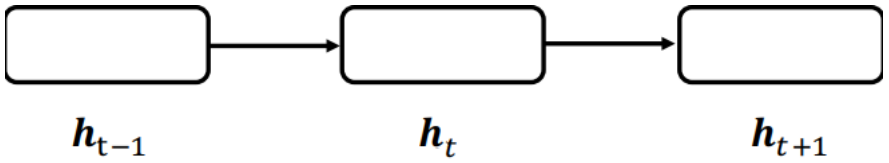
Representing Trees

- Trees structures are useful for representing syntax, semantics, etc
- **Tree LSTMs**
  - constructed by extending a sequence LSTM model
  - recurrent time steps can be taken in the bottom-up direction and receive information from its subnodes recurrently
  - top tree node can contain features over the entire tree structure
  - multiple predecessors in a tree LSTM model

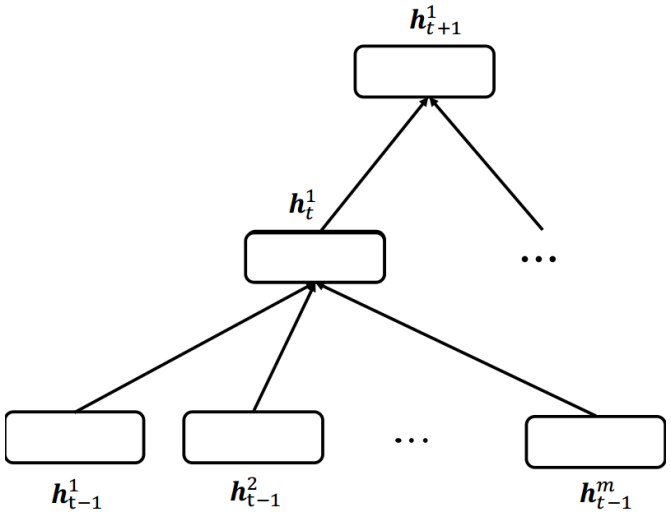
Representing Trees

WestlakeNLP

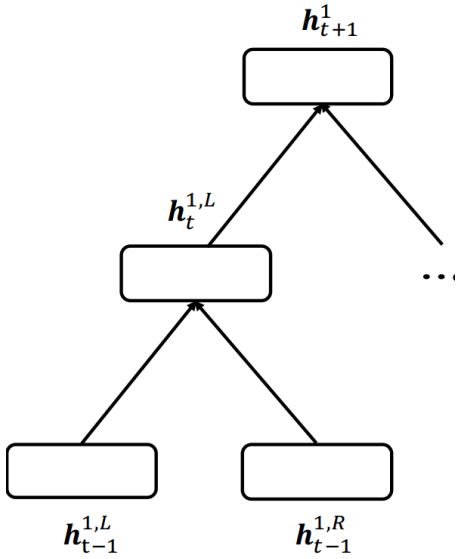
Figures of **Tree LSTMs**



(a) sequence LSTM



(b) Child-sum tree LSTM



(c) Binary tree LSTM

Sequence (a) and tree LSTMs (b and c).

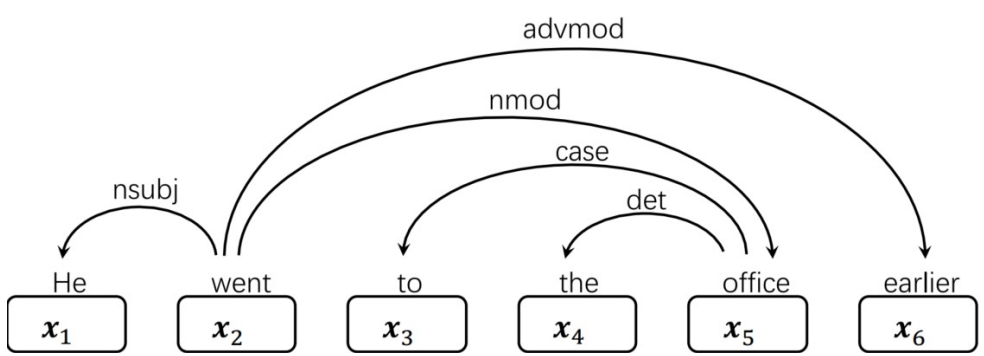
Contents

WestlakeNLP

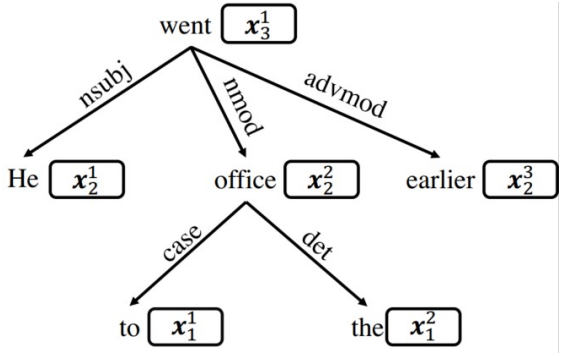
Child-Sum Tree LSTM

WestlakeNLP

- Representing arbitrary trees through turning multiple child nodes into one by summing up their hidden states
- A bottom-up recurrent computation of hidden states, and the input is rearranged hierarchically from the root
- The values of hidden nodes are calculated layer by layer



Word order



Hierarchy

Child-Sum Tree LSTM

WestlakeNLP

Notations

$\boldsymbol{X_{1:n}} = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$: embedding vectors of an input sentence

$\mathbf{h}_t$ ($t \in [1, \ldots, n]$): hidden state vectors of the input

$\mathbf{x}_t^i$: word embedding vector indexed in the bottom-up order, t is the layer index from the bottom, and i is the index within the layer

$\mathbf{h}_t^i$: hidden state vector indexed in the bottom-up order

Child-Sum Tree LSTM

WestlakeNLP

Notations

Given an embedding node $\mathbf{x}_t^i$,

- its predecessor node hidden state can be represented as

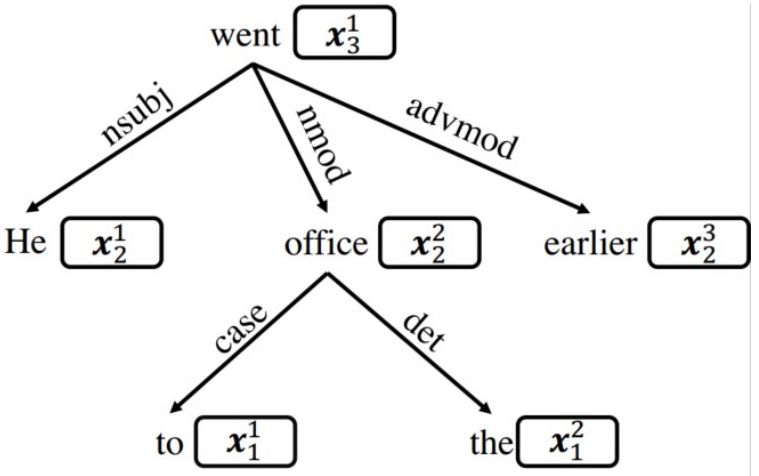
  $\mathbf{h}_{t-1}^{c(t,i,1)}, \mathbf{h}_{t-1}^{c(t,i,2)}, \ldots, \mathbf{h}_{t-1}^{c(t,i,m_t^i)}$

- its corresponding cell states can be represented as

  $\mathbf{c}_{t-1}^{c(t,i,1)}, \mathbf{c}_{t-1}^{c(t,i,2)}, \ldots, \mathbf{c}_{t-1}^{c(t,i,m_t^i)}$

where, m_t^i: the number of child nodes of $\mathbf{x}_t^i$

$c(t,i,j)$: the index of the j-th child node of $\mathbf{x}_t^i$ among nodes on the $(t-1)$-th layer



$C(2,2,1) = 1$

Child-Sum Tree LSTM

WestlakeNLP

Hidden states of all its child nodes

$\mathbf{h}_{t-1}^{c(t,i,j)}, j \in [1,\ldots,m_t^i]$ are summed up into a single hidden state $\mathbf{h}_{t-1}^{i}$ as

$$\mathbf{h}_{t-1}^{i} = \sum_{j=1}^{m_t^i} \mathbf{h}_{t-1}^{c(t,i,j)}$$

Child-Sum Tree LSTM

WestlakeNLP

Gates for Child-Sum Tree LSTM

- Given $\mathbf{h}_{t-1}^{i}$ and $\mathbf{x}_{t}^{i}$, the input gate $\mathbf{i}_{t}^{i}$ and output gate $\mathbf{o}_{t}^{i}$ are calculated as

$$\mathbf{i}_{t}^{i} = \sigma(\mathbf{W}^{ih}\mathbf{h}_{t-1}^{i} + \mathbf{W}^{ix}\mathbf{x}_{t}^{i} + \mathbf{b}^{i})$$
$$\mathbf{o}_{t}^{i} = \sigma(\mathbf{W}^{oh}\mathbf{h}_{t-1}^{i} + \mathbf{W}^{ox}\mathbf{x}_{t}^{i} + \mathbf{b}^{o}),$$

  $\mathbf{W}^{ih}$, $\mathbf{W}^{ix}$, $\mathbf{b}^{i}$, $\mathbf{W}^{oh}$, $\mathbf{W}^{ox}$ and $\mathbf{b}^{o}$ are model parameters

- For a cell state $\boldsymbol{c}_{t-1}^{c(t,i,j)}(j \in [1,\ldots,m_t^i])$, the forget gates are calculated as

$$\mathbf{f}_{t}^{i,j} = \sigma(\mathbf{W}^{fh}\mathbf{h}_{t-1}^{c(t,i,j)} + \mathbf{W}^{fx}\mathbf{x}_{t}^{i} + \mathbf{b}^{f}),$$

  $\mathbf{W}^{fh}$, $\mathbf{W}^{fx}$ and $\mathbf{b}^{f}$ are model parameters

Child-Sum Tree LSTM

WestlakeNLP

Calculating the cell states $\mathbf{c}_t$ and the hidden state $\mathbf{h}_t^i$

- The cell state $\mathbf{c}_t$ is calculated as

$$\mathbf{g}_t^i = \tanh(\mathbf{W}^{gh}\mathbf{h}_{t-1}^i + \mathbf{W}^{gx}\mathbf{x}_t^i + \mathbf{b}^g)$$

$$\mathbf{c}_t^i = \mathbf{i}_t^i \otimes \mathbf{g}_t + \sum_{j=1}^{m_t^i} \mathbf{f}_t^{i,j} \otimes \mathbf{c}_{t-1}^{c(t,i,j)},$$

$\mathbf{W}^{gh}$, $\mathbf{W}^{gx}$ and $\mathbf{b}^g$: model parameters

$\mathbf{g}_t^i$: a new cell state with the input $\mathbf{x}_t^i$ being considered

$\otimes$: Hadamard product

- $\mathbf{h}_t^i$ can be calculated as $\mathbf{h}_t^i = \mathbf{o}_t^i \otimes \tanh(\mathbf{c}_t^i)$

Contents

WestlakeNLP

Binary Tree LSTM

WestlakeNLP

- Binary tree: each node has at most two child nodes
- The hidden state of each child node to be considered separately
- More fine-grained in computing gate and cell values
- Goal: calculating a hidden vector $\mathbf{h}_t^i$ for each node in a tree LSTM. (t is the bottom-up layer index and i is the in-layer node index)

Binary Tree LSTM

WestlakeNLP

Notations

x_t^i: word embedding vector indexed in the bottom-up order, t is the layer index from the bottom, and i is the index within the layer

$\mathbf{h}_t^i$: hidden state vector indexed in the bottom-up order

$\mathbf{c}_t^i$: cell state vector indexed in the bottom-up order

$\mathbf{h}_{t-1}^{\mathrm{b}(t,i,L)}, \mathbf{h}_{t-1}^{\mathrm{b}(t,i,R)}$: hidden state values of left and right child of x_t^i

$\mathbf{c}_{t-1}^{\mathrm{b}(t,i,L)}, \mathbf{c}_{t-1}^{\mathrm{b}(t,i,R)}$: cell values of left and right child of x_t^i

$b(t,i,L)$, $b(t,i,R)$: the index of the left and right child of x_t^i among nodes on the $(t-1)$-th layer

Binary Tree LSTM

WestlakeNLP

For binary tree LSTM, recurrent LSTM steps follow sequential LSTM cell computation, but differentiating the two predecessor states of each node

The input gate $\mathbf{i}_t^i$ and two forget gates $\mathbf{f}_t^{i,L}$ and $\mathbf{f}_t^{i,R}$ are computed as follows:

$$\mathbf{i}_t^i = \sigma(\mathbf{W}_L^{ih}\mathbf{h}_{t-1}^{b(t,i,L)} + \mathbf{W}_R^{ih}\mathbf{h}_{t-1}^{b(t,i,R)} + \mathbf{W}_L^{ic}\mathbf{c}_{t-1}^{b(t,i,L)} + \mathbf{W}_R^{ic}\mathbf{c}_{t-1}^{b(t,i,R)} + \mathbf{b}^i)$$
$$\mathbf{f}_t^{i,L} = \sigma(\mathbf{W}_L^{f_l h}\mathbf{h}_{t-1}^{b(t,i,L)} + \mathbf{W}_R^{f_l h}\mathbf{h}_{t-1}^{b(t,i,R)} + \mathbf{W}_L^{f_l c}\mathbf{c}_{t-1}^{b(t,i,L)} + \mathbf{W}_R^{f_l c}\mathbf{c}_{t-1}^{b(t,i,R)} + \mathbf{b}^{f_l})$$
$$\mathbf{f}_t^{i,R} = \sigma(\mathbf{W}_L^{f_r h}\mathbf{h}_{t-1}^{b(t,i,L)} + \mathbf{W}_R^{f_r h}\mathbf{h}_{t-1}^{b(t,i,R)} + \mathbf{W}_L^{f_r c}\mathbf{c}_{t-1}^{b(t,i,L)} + \mathbf{W}_R^{f_r c}\mathbf{c}_{t-1}^{b(t,i,R)} + \mathbf{b}^{f_r})$$

$\mathbf{W}_L^{ih}, \mathbf{W}_R^{ih}, \mathbf{W}_L^{ic}, \mathbf{W}_R^{ic}, \mathbf{b}^i, \mathbf{W}_L^{f_l h}, \mathbf{W}_R^{f_l h}, \mathbf{W}_L^{f_l c}, \mathbf{W}_R^{f_l c}, \mathbf{b}^{f_l}, \mathbf{W}_L^{f_r h}, \mathbf{W}_R^{f_r h}, \mathbf{W}_L^{f_r c}, \mathbf{W}_R^{f_r c}, \mathbf{b}^{f_l}$ and $\mathbf{b}^{f_r}$ are model parameters

Binary Tree LSTM

WestlakeNLP

The cell state and hidden state values are computed as follows:

$$
\begin{aligned}
\mathbf{g}_t^i &= \tanh(\mathbf{W}_L^{gh}\mathbf{h}_{t-1}^{b(t,i,L)} + \mathbf{W}_R^{gh}\mathbf{h}_{t-1}^{b(t,i,R)} + \mathbf{b}^g) \\
\mathbf{c}_t^i &= \mathbf{i}_t^i \otimes \mathbf{g}_t^i + \mathbf{f}_t^{i,R} \otimes \mathbf{c}_{t-1}^{b(t,i,R)} + \mathbf{f}_t^{i,L} \otimes \mathbf{c}_{t-1}^{b(t,i,L)} \\
\mathbf{o}_t^i &= \sigma(\mathbf{W}_L^{oh}\mathbf{h}_{t-1}^{b(t,i,L)} + \mathbf{W}_R^{oh}\mathbf{h}_{t-1}^{b(t,i,R)} + \mathbf{W}^{oc}\mathbf{c}_t^i + \mathbf{b}^o) \\
\mathbf{h}_t^i &= \mathbf{o}_t \otimes \tanh(\mathbf{c}_t^i),
\end{aligned}
$$

$\mathbf{W}_L^{gh}, \mathbf{W}_R^{gh}, \mathbf{b}^g, \mathbf{W}_L^{oh}, \mathbf{W}_R^{oh}, \mathbf{W}^{oc}$ and $\mathbf{b}^o$ are model parameters

Tree LSTM Features and Sequence LSTM Features

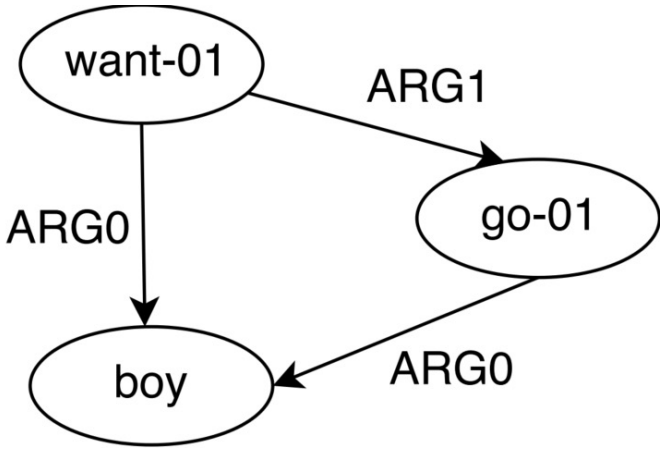
WestlakeNLP

- Difference between Tree LSTM and Sequence LSTM
  - Sequence LSTM: Integrating local word-level features into hidden representations that reflect a sentence-level context
  - Tree LSTM: Control the process of information integration, whereby syntactically correlated words are integrated before unrelated words, stronger in capturing long-range syntactic dependencies
- The **representation power** of tree LSTMs can be further combined with that of sequence LSTMs by stacking a tree LSTM on top of a sequence LSTM

Tree LSTMs and DAG LSTM

WestlakeNLP

- Directed Acyclic graph (DAG)



- Extension of tree LSTM into Lattice LSTM,
- More than one predecessors and successors.

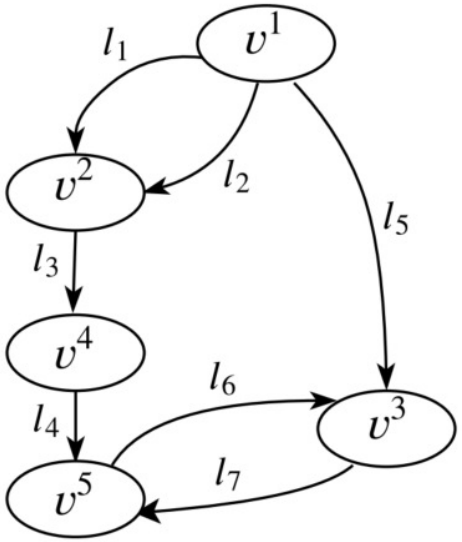
Contents

Representing Graphs

WestlakeNLP

Examples of general graph structures

- Semantic graph
  - Cyclic structure, which causes difficulty in finding a natural order of nodes in a graph
  - Hard to define recurrent time steps for calculating hidden states



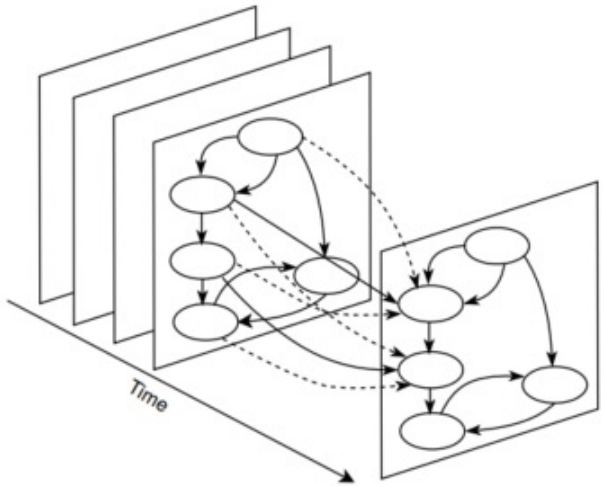
Cyclic graph

Representing Graphs

WestlakeNLP

To calculate a hidden state for representing a node in a large graph-level context:

- graph nodes can be made independent of a node order
- each node can collect information from its neighbors recurrently



Recurrent graph state update

Representing Graphs

WestlakeNLP

To calculate a hidden state for representing a node in a large graph-level context:

- time steps can be taken in a direction that is orthogonal to the graph edges
- View as a sequence of "snapshots‘’ of the graph structure
  - Each "snapshot‘’ represents a recurrent time step
  - At each time step, the hidden state is updated by collecting information from the hidden states of itself and its neighbors in the previous time step.
  - Viewed as a *message passing* time step, where each node collects information from its neighbors as a *message* for updating its own state.

Representing Graphs

WestlakeNLP

Notations

$\{V, E\}$: the graph

$V = \{v_1, v_2, \ldots v_{|V|}\}$: nodes in the graph

$E = \{e_1, e_2, \ldots, e_{|E|}\}$: edges in the graph

$e_i = (v_i^1, l_i, v_i^2)$: the connection of two nodes v_i^1 and v_i^2 with an edge labelled l_i ($i \in [1, \ldots, |E|]$)

For **directed graphs**, we assume that e_i points from v_i^1 to v_i^2

Representing Graphs

WestlakeNLP

Graph neural network (GNN)

- Assigns an initial hidden state vector $\mathbf{h}_0^i$ for each v_i ($i \in [1, \ldots, |V|]$), and then recurrently calculates $\mathbf{h}_1^i, \mathbf{h}_2^i, \ldots, \mathbf{h}_T^i$ as the hidden state for representing v_i
- $\mathbf{h}_t^i$ represents the hidden state for node i at step t
- The total number of time steps T can be decided empirically according to a task that uses the representation

Contents

WestlakeNLP

Graph Recurrent Neural network (GRN)

WestlakeNLP

Graph recurrent neural network (GRN)

- Calculating the hidden states $\mathbf{h}_1^i, \mathbf{h}_2^i, \ldots, \mathbf{h}_T^i$ for a node v_i in a recurrent process
- Given an aggregated previous state $\mathbf{m}_{t-1}^i$ and a current input $\mathbf{x}^i$, the hidden state $\mathbf{h}_t^i (t \in [1, \ldots, T])$ is calculated as:

$$\mathbf{h}_t^i = \text{LSTM\_STEP}(\mathbf{m}_{t-1}^i, \mathbf{x}^i),$$

$\mathbf{m}_{t-1}^i$: the aggregation vector of previous hidden states of v^i

$\mathbf{x}^i$: the aggregation vector of the input representation over the neighbors of v^i

Graph Recurrent Neural network (GRN)

WestlakeNLP

Graph recurrent neural network (GRN)

- The aggregated state $\mathbf{m}_t^i$: message received by v_i at time t
- For undirected graphs, or disregarding edge directions in directed graphs, given neighbours of node v_i as $\Omega(i)$, $\mathbf{m}_{t-1}^i$ can be represented as:

$$\mathbf{m}_{t-1}^i = \sum_{k \in \Omega(i)} \mathbf{h}_{t-1}^k$$

Graph Recurrent Neural network (GRN)

WestlakeNLP

Graph recurrent neural network (GRN)

x^i represents the inherent natures (integrating both node and edge information) of the graph node v_i can be defined as:

$$\mathbf{x}^i = \sum_{k \in \Omega(i)} (\mathbf{W}^x (emb(v_i) \oplus emb^e(l(i,k)) \oplus emb(v_k)) + \mathbf{b}^x),$$

emb: the embedding for a node

emb^e: the embedding for an edge

$l(i,k)$: edge label between v_i and v_k

$\mathbf{W}^x$ and $\mathbf{b}^x$: model parameters

Graph Recurrent Neural network (GRN)

WestlakeNLP

Differentiating edge directions

For directed graphs, neighbor nodes can be grouped by the edge direction for more fine-grained representation.

$\mathbf{m}_{t-1}^{i}$ for v_i can be calculated as:

$$\mathbf{m}_{t-1}^{i\uparrow} = \sum_{k \in \Omega_{\uparrow}(i)} \mathbf{h}_{t-1}^{k}$$

$$\mathbf{m}_{t-1}^{i\downarrow} = \sum_{k \in \Omega_{\downarrow}(i)} \mathbf{h}_{t-1}^{k}$$

$$\mathbf{m}_{t-1}^{i} = \mathbf{m}_{t-1}^{i\uparrow} \oplus \mathbf{m}_{t-1}^{i\downarrow}$$

$\Omega_{\uparrow}(i)$ and $\Omega_{\downarrow}(i)$: all incoming and outcoming neighbours, respectively
$\mathbf{m}_{t-1}^{i\uparrow}$ and $\mathbf{m}_{t-1}^{i\downarrow}$: previous states from neighbors with incoming and outgoing edges
$\mathbf{m}_{t-1}^{i}$: the concatenation of $\mathbf{m}_{t-1}^{i\uparrow}$ and $\mathbf{m}_{t-1}^{i\downarrow}$

Graph Recurrent Neural network (GRN)

WestlakeNLP

Differentiating edge directions

$\mathbf{x}_t^i$ of directed graphs can be defined by combining information in both edge directions

$$\mathbf{x}_t^{i\uparrow} = \sum_{k \in \Omega_\uparrow(i)} \left(\mathbf{W}_{x\uparrow} \left(emb(v_k) \oplus emb\left(l(i,k)\right) \oplus emb(v_i) \right) + \mathbf{b}_{x\uparrow} \right)$$

$$\mathbf{x}_t^{i\downarrow} = \sum_{k \in \Omega_\downarrow(i)} \left(\mathbf{W}_{x\downarrow} \left(emb(v_k) \oplus emb\left(l(i,k)\right) \oplus emb(v_i) \right) + \mathbf{b}_{x\downarrow} \right)$$

$$\mathbf{x}_t^i = \mathbf{x}_t^{i\uparrow} \oplus \mathbf{x}_t^{i\downarrow},$$

$\mathbf{W}_{x\uparrow}, \mathbf{b}_{x\uparrow}, \mathbf{W}_{x\downarrow}$ and $\mathbf{b}_{x\downarrow}$: model parameters

$l(k,i)$: the label of edge from v_k to v_i

Contents

Graph Convolutional Neural network (GCN)

WestlakeNLP

Graph convolutional neural network (GCN)

- GCN uses a convolution function to calculate $\mathbf{h}_t^i$ based on $\mathbf{h}_{t-1}^i$
- Using the same equations with GRN for calculating $\mathbf{m}_t^i$ and $\mathbf{x}_t^i$
- For updating node states, GCN uses the convolutional function as follows:

$$\mathbf{h}_t^i = \sigma(\mathbf{W}^m \mathbf{m}_{t-1}^i + \mathbf{W}^x \mathbf{x}_t^i + \mathbf{b}),$$

$\mathbf{W}^m, \mathbf{W}^x$ and $\mathbf{b}$ are model parameters

Graph Convolutional Neural network (GCN)

WestlakeNLP

Different edge labels

A variant of GCN collects information separately from different neighbors, using different weights for edges with different labels.

Donating **edge label** between v_i and v_k as $l(i,k)$ and **edge direction** between v_i and v_k as $dir(i,k)$, a GCN can be redefined as

$$\mathbf{h}_t^i = \sigma\Big(\sum_{k\in\Omega(i)}\big(\mathbf{W}^m_{l(i,k),dir(i,k)}\mathbf{h}^k_{t-1} + \mathbf{W}^x_{l(i,k),dir(i,k)}\mathbf{x}^k_t + \mathbf{b}_{l(i,k),dir(i,k)}\big)\Big)$$

$$\mathbf{x}^k_t = \Big(emb(v_k) \oplus emb\big(e(i,k)\big) \oplus emb(v_i)\Big),$$

$\mathbf{W}^m_{l(i,k),dir(i,k)}$: $|L|\times 2$ sets of model parameters to replace a single $\mathbf{W}^m$. Similar extension to $\mathbf{W}^x$ and $\mathbf{b}$. L : the set of edge labels

Graph Convolutional Neural network (GCN)

WestlakeNLP

Adding Gates

Another variant of GCN applies **gates** to control the amount of information passed from each $\mathbf{h}^k (k \in \Omega(i))$ to $\mathbf{h}^i$

The value of a gate $\mathbf{g}_t^{i,k}$ can be defined as:

$$\mathbf{g}_t^{i,k} = \sigma(\mathbf{h}_{t-1}^k \mathbf{W}_{l(i,k),dir(i,k)}^g + \mathbf{b}_{l(i,k),dir(i,k)}^g),$$

$\mathbf{W}_{l(i,k),dir(i,k)}^g$ and $\mathbf{b}_{l(i,k),dir(i,k)}^g$ are $|L|\times 2$ sets of model parameters

The **gate** can be used for updating node states as follows

$$\mathbf{h}_t^i = \sigma\Big(\sum_{k\in\Omega(i)} \mathbf{g}_t^{i,k} \otimes \big(\mathbf{W}_{l(i,k),dir(i,k)}^m \mathbf{h}_{t-1}^k + \mathbf{W}_{l(i,k),dir(i,k)}^x \mathbf{x}_t^k + \mathbf{b}_{l(i,k),dir(i,k)} \big) \Big)$$

Contents

Graph Attention Neural network (GAT)

WestlakeNLP

Using attention functions for aggregating information from neighbor states at each recurrent step

$\mathbf{h}_t^i$ for v_t at step t is defined as follows:

$$\mathbf{h}_t^i = \sum_{k \in \Omega(i)} \alpha_{ik} \, \mathbf{h}_{t-1}^k$$

α_{ik}: normalising a set of attention scores, each calculated using the previous hidden states $\mathbf{h}_{t-1}^i$ and $\mathbf{h}_{t-1}^k$ as follows:

$$s_{ik} = \sigma\left(\mathbf{W}(\mathbf{h}_{t-1}^i \oplus \mathbf{h}_{t-1}^k)\right)$$

$$\alpha_{ik} = \frac{\exp(s_{ik})}{\sum_{k' \in \Omega(i)} \exp(s_{ik'})},$$

W: a model parameter

Graph Attention Neural network (GAT)

WestlakeNLP

- GATs also have variants
- Graph Transformer is built on Transformer.

Contents

Feature Aggregation

WestlakeNLP

- GNNs calculate **a hidden state** for each node in a graph structure
- Adding one aggregation layer (pooling or attention aggregation) on top of the final $\boldsymbol{h}_i$ $(i \in [1, \ldots, |V|])$ to obtain a single **vector representation of the whole graph**

Contents

Analysing Representation

WestlakeNLP

The neural representation vector **h**

Dynamically computed low-dimensional dense

- **Pros**
  - contain automatic combinations of input features
  - capturing syntactic and semantic information
- **Cons**
  - not easily interpretable

Two indirect ways to analyse learned representation vectors

- Visualisation
- Probing tasks
- Ablation

Analysing Representation

WestlakeNLP

Visualisation

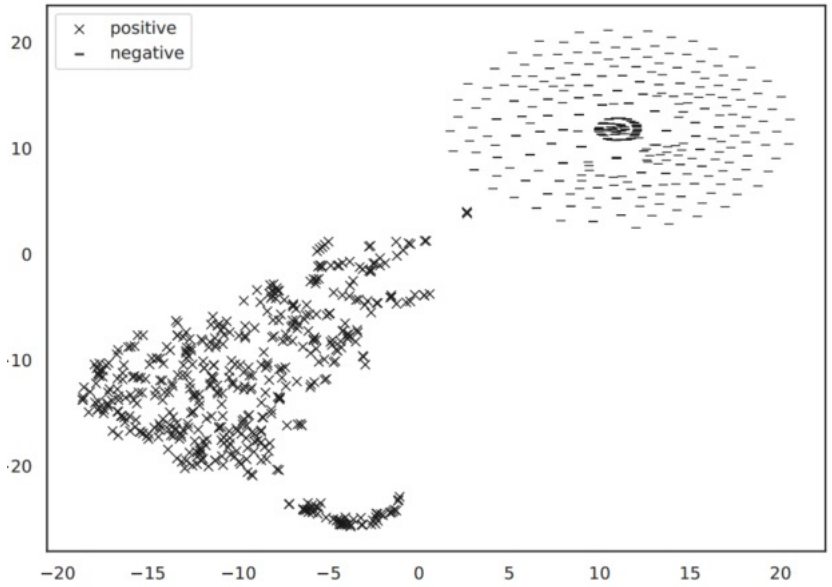
- Projecting hidden representations into a two-dimensional figure to better understand their correlations
- Preserving the distance correlation between vectors to gain knowledge about the characteristics of the representation vectors
- A useful tool: **t-distributed stochastic neighbor embedding** (**t-SNE**)

Analysing Representation

WestlakeNLP

t-distributed stochastic neighbor embedding (t-SNE)

A non-linear dimensionality reduction technique that aims to **preserve the distance correlation between vectors** in the original high-dimensional vector space and then projected to two-dimensional space.



An example of t-SNE visualisation of positive and negative documents.

Analysing Representation

WestlakeNLP

Probing tasks

- Auxiliary tasks that predict the features that we expect a learned representation to capture.
- Using a set of additional output layers.
- Procedures
  - given a set of documents with gold-standard outputs
  - run the representation model and dump the vector representation
  - train a very simple classification model, and treat the probed task as the output
  - the more accurate the trained simple model is, the more confident we are that the representation vectors contain relevant information

Ablation

WestlakeNLP

- Remove a vector from a set of hidden states.
- Check output.

Contents

More on Neural Network Training

WestlakeNLP

- Optimisation technique: A key to successful representation learning especially for neural network training
- Simple methods such as SGD may not give the best optimisation towards a training objective because the neural network structure becomes increasingly deep and complex
- This section will list more alternatives for optimisation

Contents

WestlakeNLP

AdaGrad

WestlakeNLP

- AdaGrad: an optimisation algorithm that adaptively sets the learning rate for each parameter based on the gradient
- Notations
  - Θ: model parameters
  - **g**: the corresponding set of gradients
- For each parameter $\theta_i \in \Theta (i \in [1, \dots, |\Theta|])$, AdaGrad maintains an accumulated squared gradient sg_i from the start of training to estimate the per-parameter learning rate.
- The learning rate η_i for θ_i is inversely proportional to the root of sg_i

More on Neural Network Training

WestlakeNLP

- The update rules of AdaGrad can be written as:

$$\mathbf{g}_t = \frac{\partial L(\Theta_{t-1})}{\partial \Theta_{t-1}}$$

$$sg_{t,i} = sg_{t-1,i} + g_{t,i}^2$$

$$\theta_{t,i} = \theta_{t-1,i} - \frac{\eta}{\sqrt{sg_{t,i} + \epsilon}} g_{t,i}$$

L: loss

ϵ: a hyper-parameter for numerical stability

t: the time step number in parameter update

$sg_{t,i}$: the sum of squares of the gradient with respect to θ_i

- Common hyper-parameter settings:
  - $\epsilon = 1e^{-8}$
  - $\eta = 0.01$

Contents

RMSProp

WestlakeNLP

- Problems for AdaGrad
  - the learning rate decreases monotonically and aggressively, which can lead to early and suboptimal convergence
  - sensitive to initial gradients
- RMSProp solves the problems of AdaGrad by
  - using attention to a limited history window instead of all history gradients
  - the initial gradient does not greatly affect the learning rate of future time steps

RMSProp

WestlakeNLP

- The update rules of RMSProp can be written as:

$$\mathbf{g}_t = \frac{\partial L(\Theta_{t-1})}{\partial \Theta_{t-1}}$$

$$\mathrm{E}\,|\,\mathbf{g}^2\,|_t = \rho \mathrm{E}\,|\,\mathbf{g}^2\,|_{t-1} + (1-\rho)\mathbf{g}_t^2$$

$$RMS|\mathbf{g}|_t = \sqrt{\mathbb{E}\,|\mathbf{g}^2|_t + \epsilon}$$

$$\Theta_t = \Theta_{t-1} - \frac{\eta}{RMS\,|\,\mathbf{g}\,|_t}\mathbf{g}_t$$

$\mathbb{E}|\mathbf{g}^2|_t$: the dynamic average of the squares of the gradients. ρ: a hyper-parameter controlling the percentage of the previous average and the current gradient

- The remaining updating rules are the same as AdaGrad
- Common hyper-parameter settings:
  - $\rho = 0.9 \quad \eta = 0.001$

Contents

AdaDelta

WestlakeNLP

- dealing with the learning rate decay problem of AdaGrad, with an exponentially running average of the square of history gradients
- replacing manual selection of the initial learning rate η with an estimation of $\Delta\Theta$ at the t-th timestep
- The key idea is to make the parameter update $\Delta\Theta$ proportional to the parameter Θ itself

AdaDelta

WestlakeNLP

- The update rules of AdaDelta can be written as:

$$\mathbb{E}\,|\Delta\Theta^2|_t = \rho\,\mathbb{E}\,|\Delta\Theta^2|_{t-1} + (1-\rho)\Delta\Theta_t^2$$

$$RMS|\Delta\Theta|_t = \sqrt{\mathbb{E}\,|\Delta\Theta^2|_t + \epsilon}.$$

$\Delta\Theta$: the parameter change

$\mathbb{E}|\Delta\Theta^2|$: the exponential running averaging of the squares of the parameter change

- $RMS|\Delta\Theta|_t$ remains **unkown** before calculating $\Delta\Theta$
- Therefore, AdaDelta approximate $RMS|\Delta\Theta|_t$ by assuming $RMS(\cdot)$ function is locally smooth

AdaDelta

WestlakeNLP

- The update rules for $\Delta\Theta_t$ can be written as:

$$\Delta\Theta_t = -\frac{RMS|\Delta\Theta|_{t-1}}{RMS|\mathbf{g}|_t}\mathbf{g}_t$$

$RMS|\Delta\Theta|_{t-1}$: an acceleration term, summarising the history parameter update within a recent window

- Common hyper-parameter settings:
  - ρ: 0.9
  - ϵ: $1e^{-6}$

Contents

WestlakeNLP

Adam

WestlakeNLP

- integrates the ideas of momentum SGD and RMSProp by maintaining the exponentially running averages of both the first order **moment** and the second order **moment**
- moment: a mathematical tool for quantitative description of the shape of the gradient function
  - first order moment: records the moving average of history gradients
  - second order moment: accumulates the moving average of history squared gradients

Adam

WestlakeNLP

- The two gradient estimations are defined as:

$$\mathbf{g}_t = \frac{\partial L(\Theta_{t-1})}{\partial \Theta_{t-1}}$$

$$\mathbf{v}_t = \beta_1 \mathbf{v}_{t-1} + (1-\beta_1)\mathbf{g}_t$$

$$\mathrm{E}\,|\,\mathbf{g}^2\,|_t = \beta_2 \mathrm{E}\,|\,\mathbf{g}^2\,|_{t-1} + (1-\beta_2)\mathbf{g}_t^2$$

- $\mathbf{v}$: a first order moment estimation, acting as the momentum
- $\mathbb{E}|\mathbf{g}^2|$: a second order moment estimation, representing the running expectation of the squares of the gradients as in RMSProp.
- β_1 and β_2 : hyper-parameters, which are both recommended to be set to close 1.

Adam

WestlakeNLP

- The initial values of $\mathbf{v}$ and $\mathbf{g}$ are both zeroes
- At time step t, $\mathbf{v}_t$ (a weighted sum of gradients within time step t) is given by

$$
\begin{aligned}
\mathbf{v}_1 &= \beta_1 \mathbf{v}_0 + (1-\beta_1)\mathbf{g}_1 = (1-\beta_1)\mathbf{g}_1 \\
\mathbf{v}_2 &= \beta_1 \mathbf{v}_1 + (1-\beta_1)\mathbf{g}_2 = \beta_1(1-\beta_1)\mathbf{g}_1 + (1-\beta_1)\mathbf{g}_2 \\
&= (1-\beta_1)(\beta_1 \mathbf{g}_1 + \mathbf{g}_2) \\
&\cdots \\
\mathbf{v}_t &= (1-\beta_1)(\beta_1^{t-1}\mathbf{g}_1 + \beta_1^{t-2}\mathbf{g}_2 + \cdots + \mathbf{g}_t)
\end{aligned}
$$

Adam

WestlakeNLP

- b_t, which is the sum of the weights of the gradients $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_t$, is given by

$$b_t = (1-\beta_1)(\beta_1^{t-1} + \beta_1^{t-2} + \cdots + 1) = (1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}$$

$$= \sum_{i=1}^{t}\beta_1^{t-i} - \sum_{i=1}^{t}\beta_1^{t+1-i}$$

$$= 1-\beta_1^t,$$

- b_t is not equal to 1, which indicates that Adam is **biased towards zero parameter** update in the beginning steps

Adam

WestlakeNLP

To remedy these biases, Adam uses bias-corrected estimations

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_1^t}$$

The bias-corrected estimations for the second order moment is

$$\widehat{\mathbb{E}}|\mathbf{g}^2|_t = \frac{\mathbb{E}|\mathbf{g}^2|_t}{1-\beta_2^t}$$

The final update rule for Adam applied to θ_t is

$$\Theta_t = \Theta_{t-1} - \frac{\eta}{\sqrt{\widehat{\mathbb{E}}|\mathbf{g}^2|_t + \epsilon}}\hat{\mathbf{v}}_t$$

Common hyper-parameter settings:

- ϵ:$1e^{-8}$
- η:$1e^{-3}$

Contents

Choosing a Training Method

WestlakeNLP

- The performance of these adaptive gradient optimisers can vary with different datasets and hyper-parameter choices
- The choice of the optimiser itself can be viewed as a hyper-parameter
  - Adam
    - the most popular choice of the adaptive gradient optimisers
    - converges much faster than SGD with momentum
  - SGD
    - obtain good or even better performances with careful learning rate decay compared to Adam

Summary

WestlakeNLP

- Recurrent Neural Network and LSTM
- Attention and Self Attention network
- Tree LSTMs
- Graph Neural Network (GCN, GRN, GAT)
- Explainability of neural representations
- SGD extensions.