Week 10: Recurrent Neural Networks

Instructor: Ruixuan Wang
wangruix5@mail.sysu.edu.cn

School of Data and Computer Science
Sun Yat-Sen University

28 April (for 02 May), 2019
1. Word2vec & language modelling
2. RNN & LSTM
3. RNN app examples
4. RNN structures
Motivation: natural language processing

- Sentence or document classification (topic, sentiment)
- Topic modelling
- Translation
- Chatbots, dialogue system, assistant
- Summarization
Word representation

- Words are originally represented as 1-hot vectors
Word representation

- Words are originally represented as 1-hot vectors
- Large vocabulary of possible words
- Use of word **embeddings** as inputs in deep NLP models
- Word embeddings usually have dimensions 50 - 300
Word representation

- Words are originally represented as 1-hot vectors
- Large vocabulary of possible words
- Use of word **embeddings** as inputs in deep NLP models
- Word embeddings usually have dimensions 50 - 300
- Then how to obtain such embedding?
Word2vec: skip-gram model

- Given central word, predict occurrence of other words in context

**Source Text**

- The **quick** brown fox jumps over the lazy dog.
- The **quick** brown **fox** jumps over the lazy dog.
- The **quick** **brown** **fox** jumps **over** the lazy dog.
- The **quick** brown **fox** jumps over the lazy dog.

**Training Samples**

- (the, quick)
- (the, brown)
- (quick, the)
- (quick, brown)
- (quick, fox)
- (brown, the)
- (brown, quick)
- (brown, fox)
- (brown, jumps)
- (fox, quick)
- (fox, brown)
- (fox, jumps)
- (fox, over)
Skip-gram model: a simple 2-layer neural network

- **Input Vector:**
  - A '1' in the position corresponding to the word “ants”
  - 10,000 positions

- **Hidden Layer:**
  - 300 neurons

- **Output Layer:**
  - Softmax Classifier
  - Probability that the word at a randomly chosen, nearby position is “abandon”
  - "ability"
  - "able"
  - "zone"

- **RNN & LSTM**
- **RNN app examples**
- **RNN structures**
- **Word2vec & language modelling**
Two words having similar contexts: ‘intelligent’ and ‘smart’, ‘ant’ and ‘ants’, etc.
Skip-gram model (cont’)

- Two words having similar contexts: ‘intelligent’ and ‘smart’, ‘ant’ and ‘ants’, etc.
- If two words have very similar contexts, then skip-gram model needs to output similar results.
Skip-gram model (cont’)

- Two words having similar contexts: ‘intelligent’ and ‘smart’, ‘ant’ and ‘ants’, etc.
- If two words have very similar contexts, then skip-gram model needs to output similar results.
- Then the skip-gram network is motivated to learn similar word vectors (at hidden layer) for these similar words!
Skip-gram model: vector space is semantic
Word2vec: continuous bag of words (CBOW) model

- ‘Reverse’ of skip-gram
- $C$ context words as input
- Center word as output
- Hidden layer: average over $C$ embeddings, hence ‘bag of words’
- Training: again with cross-entropy loss
Word2vec is just for word representation.

How to capture meaning of sentence/paragraph?
Word2vec is just for word representation.

How to capture meaning of sentence/paragraph?

We need consider order of words in text!
Language models

Language models: assign a probability to a sequence of words, such that plausible sequences have higher probabilities, e.g.,

- $p(‘I \text{ like apples’}) > p(‘I \text{ sit apples’})$
- $p(‘I \text{ like apples’}) > p(‘like I apples’)
Language models: assign a probability to a sequence of words, such that plausible sequences have higher probabilities, e.g.,

- \( p(‘I\ like\ apples’) > p(‘I\ sit\ apples’) \)
- \( p(‘I\ like\ apples’) > p(‘like\ I\ apples’) \)

Auto-regressive modelling of sequence \((w_0, w_1, \ldots, w_n)\):

\[
p(w_0) \cdot p(w_1|w_0) \cdots p(w_n|w_{n-1}, w_{n-2}, \ldots, w_0)
\]
Language models

Language models: assign a probability to a sequence of words, such that plausible sequences have higher probabilities, e.g.,

- $p(‘I$ like apples’) > $p(‘I$ sit apples’)
- $p(‘I$ like apples’) > $p(‘like I apples’)$

Auto-regressive modelling of sequence $(w_0, w_1, \ldots, w_n)$:

$$p(w_0) \cdot p(w_1|w_0) \cdots p(w_n|w_{n-1}, w_{n-2}, \ldots, w_0)$$

- $p(\cdot)$ can be a neural network!
- $p(\cdot)$ could capture meaning of sequential information
Conditional language models for NLP problems

Translation: $p(\text{Target}|\text{Source})$

- Source (Chinese): ‘wo xi huan ping guo’
- Target (English): ‘I like apples’
- Model the output word by word:

$$p(w_0|\text{Source}) \cdot p(w_1|w_0, \text{Source}) \ldots$$
Conditional language models for NLP problems

Translation: \( p(\text{Target}|\text{Source}) \)
- Source (Chinese): ‘wo xi huan ping guo’
- Target (English): ‘I like apples’
- Model the output word by word:

\[
p(w_0|\text{Source}) \cdot p(w_1|w_0, \text{Source}) \ldots
\]

Question answering / Dialogue: \( p(\text{Answer}|\text{Question}, \text{Context}) \)
- Context:
  - ‘John puts two apples on the table.’
  - ‘Tom adds three more apples.’
  - ‘Tom leaves to study in the library.’
- Question: ‘How many apples are there?’
- Answer: ‘There are five apples.’
What neural networks can represent $p(\cdot|\cdot)$?
Recurrent neural network (RNN): basics

- Recurrent neural network: output of hidden layer at each time step is part of input to hidden layer at next time step.
Recurrent neural networks (RNN): basics

- Recurrent neural network: output of hidden layer at each time step is part of input to hidden layer at next time step.
- Unroll to process an input sequence \((x_0, x_1, x_2, \ldots)\)
Recurrent neural networks (RNN): basics

- Recurrent neural network: output of hidden layer at each time step is part of input to hidden layer at next time step.
- Unroll to process an input sequence \((x_0, x_1, x_2, \ldots)\)
- RNN is a DEEP neural network model
RNN basics

\[ h_t = g(W_h h_{t-1} + W_i x_t + b_h) \]
\[ y_t = \sigma(W_o h_t + b_o) \]

- \( g(\cdot) \): activation function, often \( \text{tanh} \); \( \sigma(\cdot) \): softmax function
- Same functions (model parameters) used at every time step!
RNN training

- Multiply same matrix at each time step during forward prop
- Inputs from many time steps ago can affect output $y_t$
- Multiply the same matrix at each time step during backprop

![Diagram of RNN training](image)

Back propagation
RNN training: gradient exploding/vanishing

- Training RNN is hard
- Similar but simpler RNN formulation:

\[
\begin{align*}
    h_t &= Wg(h_{t-1}) + W_i x_t \\
    y_t &= \sigma(W_o h_t)
\end{align*}
\]
RNN training: gradient exploding/vanishing

- Training RNN is hard
- Similar but simpler RNN formulation:

\[
\begin{align*}
    h_t &= W g(h_{t-1}) + W_i x_t \\
    y_t &= \sigma(W_o h_t)
\end{align*}
\]

- Total loss is the sum of loss over all time steps, then

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}
\]
RNN training: gradient exploding/vanishing

- Training RNN is hard
- Similar but simpler RNN formulation:

\[ h_t = Wg(h_{t-1}) + W_i x_t \]
\[ y_t = \sigma(W_o h_t) \]

- Total loss is the sum of loss over all time steps, then

\[ \frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W} \]

- With chain rule:

\[ \frac{\partial L_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W} \]
RNN training: gradient exploding/vanishing

- So far:

\[
    h_t = W g(h_{t-1}) + W_i x_t
\]

\[
    \frac{\partial L_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}
\]
RNN training: gradient exploding/vanishing

So far:

\[ h_t = W g(h_{t-1}) + W_i x_t \]

\[ \frac{\partial L_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W} \]

With chain rule again

\[ \frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T \text{diag} \left[ g'(h_{j-1}) \right] \]
RNN training: gradient exploding/vanishing

- So far:

\[ h_t = W g(h_{t-1}) + W_i x_t \]

\[ \frac{\partial L_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W} \]

- With chain rule again

\[ \frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T \text{diag} \left[ g'(h_{j-1}) \right] \]

If

\[ \| \frac{\partial h_j}{\partial h_{j-1}} \| \leq \| W^T \| \| \text{diag} \left[ g'(h_{j-1}) \right] \| \leq \beta_W \beta_h \]
RNN training: gradient exploding/vanishing

- So far:

\[ h_t = W g(h_{t-1}) + W_i x_t \]

\[ \frac{\partial L_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W} \]

- With chain rule again

\[ \frac{\partial h_t}{\partial h_k} = \Pi_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \Pi_{j=k+1}^{t} W^T \text{diag} [g'(h_{j-1})] \]

If

\[ \| \frac{\partial h_j}{\partial h_{j-1}} \| \leq \| W^T \| \| \text{diag} [g'(h_{j-1})] \| \leq \beta_W \beta_h \]

Then

\[ \| \frac{\partial h_t}{\partial h_k} \| = \Pi_{j=k+1}^{t} \| \frac{\partial h_j}{\partial h_{j-1}} \| \leq (\beta_W \beta_h)^{t-k} \]
When $\beta_W/\beta_h > 1$, 

$$\left\| \frac{\partial L_t}{\partial W} \right\| \gg 1$$
RNN training: gradient exploding/vanishing

• When $\beta_W/\beta_h > 1$, 

$$\left\| \frac{\partial L_t}{\partial W} \right\| \gg 1$$

causing gradient exploding!
RNN training: gradient exploding/vanishing

- When $\beta_W/\beta_h > 1$,

$$\left\| \frac{\partial L_t}{\partial W} \right\| \gg 1$$

causing gradient exploding!

- Trick: gradient clipping

```python
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
    grad *= (threshold / grad_norm)
```

- Gradient clipping well solved gradient exploding!
RNN training: vanishing gradient is a problem

- When $\beta_W \beta_h < 1$, vanishing $\frac{\partial h_t}{\partial h_k}$ and

\[
h_t = W g(h_{t-1}) + W_i x_t
\]

\[
\frac{\partial L_t}{\partial W_i} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \text{diag} [x_k]
\]

would cause $x_k$ from previous time step $k$ not to affect update of $W_i$ at time step $t$. 
When $\beta_W / \beta_h < 1$, vanishing $\frac{\partial h_t}{\partial h_k}$ and

$$h_t = Wg(h_{t-1}) + W_i x_t$$

$$\frac{\partial L_t}{\partial W_i} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \text{diag}[x_k]$$

would cause $x_k$ from previous time step $k$ not to affect update of $W_i$ at time step $t$.

In other words, prediction error at time step $t$ would not tell a far-away previous step $k$ to change during backprop.
RNN training: vanishing gradient is a problem

- When $\beta_W/\beta_h < 1$, vanishing $\frac{\partial h_t}{\partial h_k}$ and

  \[
  h_t = W g(h_{t-1}) + W_i x_t
  \]

  \[
  \frac{\partial L_t}{\partial W_i} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \text{diag} [x_k]
  \]

  would cause $x_k$ from previous time step $k$ not to affect update of $W_i$ at time step $t$.

- In other words, prediction error at time step $t$ would not tell a far-away previous step $k$ to change during backprop.

- Vanishing gradient makes RNN unable to capture long-term relationship between items far away from each other!
Long short-term memory (LSTM)

- LSTM as basic unit of RNN reduces gradient vanishing
Long short-term memory (LSTM)

- LSTM as basic unit of RNN reduces gradient vanishing
- ‘short-term memory’: a small amount of information
- ‘long’: information can last for a long period of time
Long short-term memory (LSTM)

- LSTM as basic unit of RNN reduces gradient vanishing
- ‘short-term memory’: a small amount of information
- ‘long’: information can last for a long period of time
- LSTM: cell, input gate, output gate, (un)forget gate
- Cell for ‘remembering’ values over arbitrary time steps, hence the word ‘memory’ in LSTM
- Gates as regulators of the flow of signals through LSTM
• Input gate: whether/how much to write to cell
• Output gate: how much to reveal cell
• Forget gate: whether/how much to erase cell
LSTM
LSTM

\[ i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \]
\[ f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \]
\[ o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \]
\[ g_t = \tanh(W_g x_t + U_g h_{t-1} + b_g) \]
\[ c_t = f_t \odot c_{t-1} + i_t \odot g_t \]
\[ h_t = o_t \odot \tanh(c_t) \]

- \( i_t, f_t, o_t \): input, forget, and output gate; \( \sigma \): sigmoid function
- \( g_t \): new signal to update cell
- \( c_t \): updated cell; \( h_t \): new hidden state
- Well chosen activation function (tanh) is critical
- Three times more parameters than RNN
LSTM

- An alternative diagram representation of LSTM

\[
\begin{align*}
(i) &= \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \\
(f, i, g, o) &= \begin{pmatrix} f \\ i \\ g \\ o \end{pmatrix} \\
c_t &= f \odot c_{t-1} + i \odot g \\
h_t &= o \odot \tanh(c_t)
\end{align*}
\]

Figures and content here and in the next 9 slide mainly from Stanford CS231n Lecture 10, 2017
Why LSTM can reduce gradient vanishing

- Additive path between $c_t$ and $c_{t-1}$

Backpropagation from $c_t$ to $c_{t-1}$ only elementwise multiplication by $f$, no matrix multiply by $W$

\[
\begin{pmatrix}
  i \\
  f \\
  o \\
  g
\end{pmatrix} =
\begin{pmatrix}
  \sigma \\
  \sigma \\
  \sigma \\
  \tanh
\end{pmatrix}
W
\begin{pmatrix}
  h_{t-1} \\
  x_t
\end{pmatrix}
\]

$c_t = f \odot c_{t-1} + i \odot g$

$h_t = o \odot \tanh(c_t)$
Why LSTM can reduce gradient vanishing

- Gradient signal can easily back propagate through multiple time steps (if forget gate is open)

Uninterrupted gradient flow!
Why LSTM can reduce gradient vanishing

- Gradient signal can easily back propagate through multiple time steps (if forget gate is open)
- Reminder: skip connections in ResNet

Uninterrupted gradient flow!
Gated recurrent unit (GRU)

\[ r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \]
\[ z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \]
\[ \hat{h}_t = \tanh(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h) \]
\[ h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \hat{h}_t \]

- One gate less than LSTM, so fewer parameters
- No ‘cell’, only hidden vector \( h_t \) passed to next unit
- No systematic difference between GRU and LSTM
- People tend to use LSTM more
Vanilla RNN for language modeling

- Predict next character based on previous characters

**THE SONNETS**

by William Shakespeare

From fairest creatures we desire increase,  
That thereby beauty's rose might never die,  
But as the riper should by time decease,  
His tender heir might bear his memory:  
But thou, contracted to thine own bright eyes,  
Feed'st thy light's flame with self-substantial fuel,  
Making a famine where abundance lies,  
Thyself thy foe, to thy sweet self too cruel:  
Thou art now the world's fresh ornament,  
And only herald to the gaudy spring,  
Within thine own bud hast thou thy content,  
And tender churl mak'st waste in niggarding:  
Pity the world, or else this gluton be,  
To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,  
And dig deep trenches in thy beauty's field,  
Thy youth's proud livery so grazed on now,  
Will be a tatter'd weed of small worth held;  
Then being asked, where all thy beauty lies,  
Where all the treasure of thy lusty days;  
To say, within thine own deep sunken eyes,  
Were an all-eating shame, and thriftless praise.  
How much more praise deserved thy beauty's use,  
If thou couldst answer 'This fair child of mine  
Shall sum my count, and make my old excuse,'  
Proving his beauty by succession thine!  
This were to be new made when thou art old,  
And see thy blood warm when thou feel'st it cold.
Language modeling

tyntd-iafhatawiaohrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e plia tklrgd t o idoe ns,smtt  h ne etie h,hregtrs nigtike,aoaenns lng

down

"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome coaniogenncc Phe lism thond hon at. MeiDimorotion in ther thize."

down

Aftair fall unsuch that the hall for Prince Velzonski's that me of her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort how, and Gogition is so overelical and ofter.

down

"Why do what that day," replied Natasha, and wishing to himself the fact the princess, Princess Mary was easier, fed in had oftened him. Pierre aking his soul came to the packs and drove up his father-in-law women.
Language modeling

- With latex source code: predict next character based on previous characters

The Stacks Project: open source algebraic geometry textbook

[Image of The Stacks Project]

Latex source

http://stacks.math.columbia.edu/
The stacks project is licensed under the GNU Free Documentation License
After training a RNN, generate latex doc, then render it.
Language modeling

- With C source code: predict next character based on previous characters
Language modeling

```c
static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << i))
            pipe = (in_use & UMXTHREAD_UNCCA) +
            ((count & 0x00000000fffffffff) & 0x000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}```
RNN variants in structure...

RNN can have more complex structures!
People may use different notations for RNN.

$h_t$ summarizes the sentence up to time step $t$.

Problem: for some tasks, it would be better to incorporate information from both preceding and following words.

$h_t = f(Wx_t + Vh_{t-1} + b)$

$y_t = g(Uh_t + c)$
Bidirectional RNN (BiRNN)

- Bidirectional RNN captures sequential information from both directions.
- RNN unit could be LSTM or others

\[ \hat{h}_t = f(\overrightarrow{W}x_t + \overrightarrow{V}\hat{h}_{t-1} + \overrightarrow{b}) \]
\[ \hat{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\hat{h}_{t+1} + \overleftarrow{b}) \]
\[ y_t = g(U[\hat{h}_t;\hat{h}_t] + c) \]
Deep bidirectional RNN

- Deep BiRNN: each layer passes an intermediate sequential representation to the next layer.

\[
\begin{align*}
\text{Forward: } & \quad \bar{h}_t = f(W^{(i)} h_{t-1}^{(i-1)} + V^{(i)} h_{t-1}^{(i)} + b^{(i)}) \\
\text{Backward: } & \quad \bar{h}_t = f(W^{(i)} h_{t+1}^{(i-1)} + V^{(i)} h_{t+1}^{(i)} + b^{(i)}) \\
\text{Output: } & \quad y_t = g(U^{(L)} h_t^{(L)} + c)
\end{align*}
\]
RNN outputs

- Left: e.g., sequence of words $\rightarrow$ sentiment
- Centre: e.g., machine translation
- Right: e.g., video classification for each frame
Summary

- Word2vec as input to RNN models
- Gradient clipping to reduce exploding issue
- Vanilla RNN may not well capture long-term relationships
- LSTM can capture long-term relationships
- LSTM can reduce gradient vanishing issue
- Different RNN structures/outputs for different apps

Further reading:

- Mikolov, Sutskever, Chen, Corrado, Dean, ‘Distributed representations of words and phrases and their compositionality’, NIPS, 2013