Lecture 5: Language Models and Recurrent Neural Network

Presenter: Chu Đình Đức

Lecture Plan

1. Language Models
2. Recurrent Neural Networks (RNN)

1. Language Models

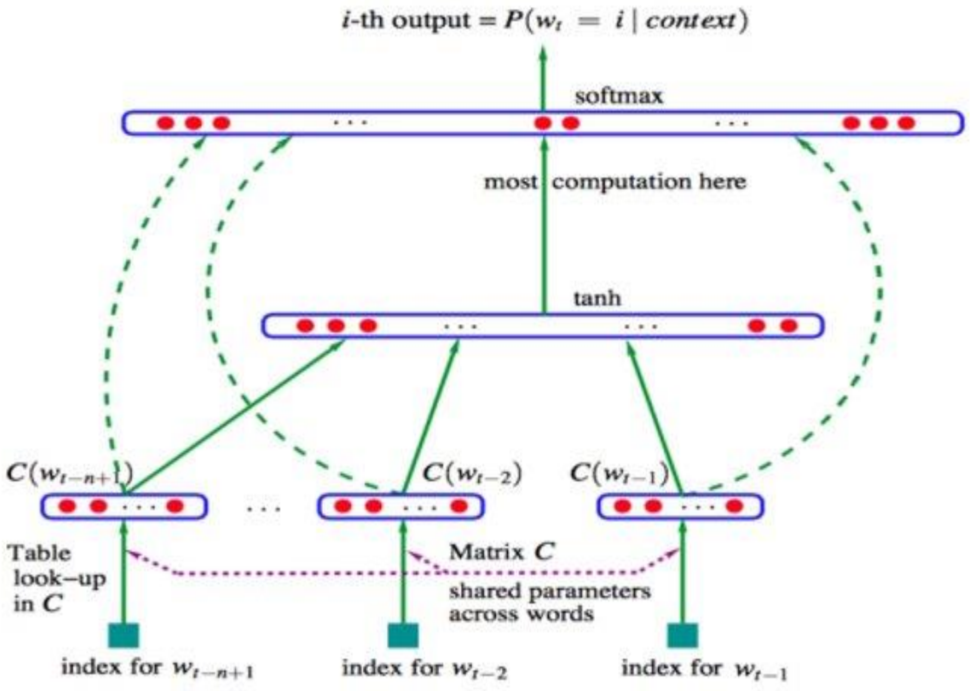
- Language models compute the probability of occurrence of a number of words in a particular sequence

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{i=m} P(w_i | w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{i=m} P(w_i | w_{i-n}, \ldots, w_{i-1})$$

- How to compute this probability?

1. Language Models

- The first deep neural network architecture model for NLP presented by Bengio et al.



$$\hat{y} = softmax(W^{(2)} tanh(W^{(1)} x + b^{(1)}) + W^{(3)} x + b^{(3)})$$

2. RNN

- Unlike the conventional translation models, RNN are capable of conditioning the model on all previous words in the corpus

$$h_t = \sigma(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]})$$

$$\hat{y}_t = softmax(W^{(S)} h_t)$$

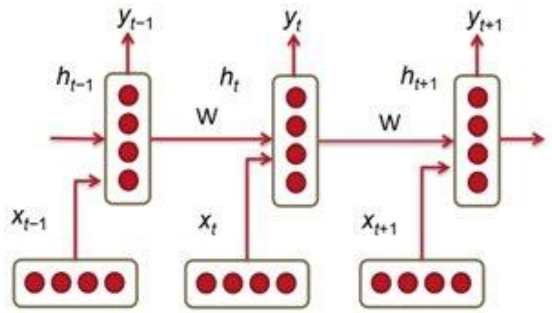


Figure 2: A Recurrent Neural Network (RNN). Three time-steps are shown.

2. RNN

- The loss function used in RNN is often the cross entropy error:

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \times log(\hat{y}_{t,j})$$

- The cross entropy error over a corpus of size T is:

$$J = -\frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{|V|} y_{t,j} \times log(\hat{y}_{t,j})$$

Lecture 6: Simple and LSTM Recurrent Neural Networks

Presenter: Chu Đình Đức

Lecture Plan

1. Vanishing Gradient & Gradient Explosion Problems
2. Deep Bidirectional RNNs
3. Application: RNN Translation Model
4. Long Short Term Memories (LSTM)
5. Gated Recurrent Units (GRU)

1. Vanishing Gradient & Gradient Explosion Problems

- Vanishing Gradient: gradients get smaller when going down to the lower layers
- Exploding Gradient: in the other case, gradients get bigger in BP progress

1. Vanishing Gradient & Gradient Explosion Problems

Solution:

- Exploding Gradient:

$$
\begin{array}{l}
\hat{g} \leftarrow \frac{\partial E}{\partial W} \\
\textbf{if } \| \hat{g} \| \geq \textit{threshold} \textbf{ then} \\
\quad \hat{g} \leftarrow \frac{\textit{threshold}}{\| \hat{g} \|} \hat{g} \\
\textbf{end if}
\end{array}
$$

- Vanishing Gradient:
  - Instead of initializing Whh randomly, start off from an identify matrix initialization
  - Use ReLU instead of the sigmoid function: gradient would flow through the neurons whose derivative is 1 without getting attenuated

2. BiRNN

- It is possible to make predictions based on future words by having RNN model read through the corpus backwards

$$\overrightarrow{h}_t = f(\overrightarrow{W} x_t + \overrightarrow{V} \overrightarrow{h}_{t-1} + \overrightarrow{b})$$

$$\overleftarrow{h}_t = f(\overleftarrow{W} x_t + \overleftarrow{V} \overleftarrow{h}_{t+1} + \overleftarrow{b})$$

$$\hat{y}_t = g(Uh_t + c) = g(U[\overrightarrow{h}_t; \overleftarrow{h}_t] + c)$$

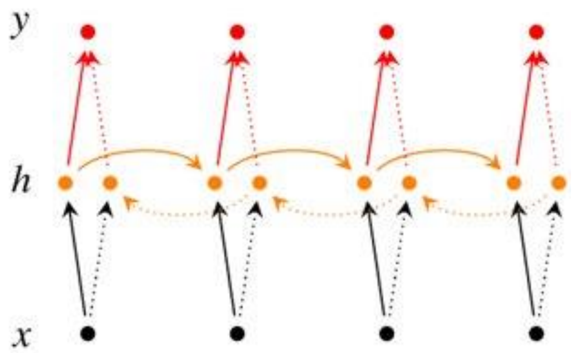


Figure 6: A bi-directional RNN model

2. Multi-layer BiRNN

$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$\hat{y}_t = g(Uh_t + c) = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

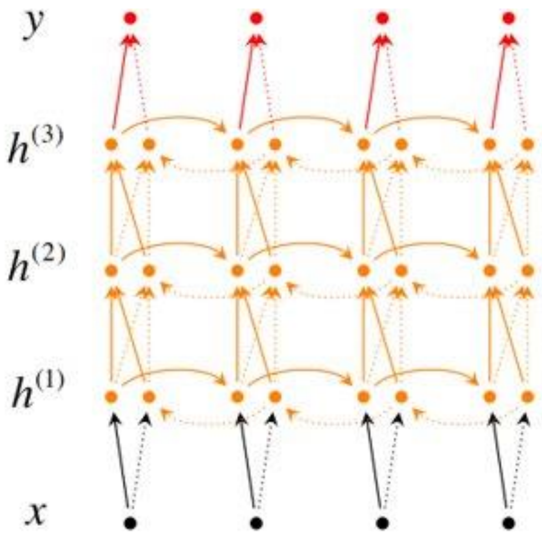


Figure 7: A deep bi-directional RNN with three RNN layers.

3. Application: RNN Translation Model

- The first three hidden layer time-steps encode the German language words into some language word features (h3)
- The last two time-steps decode h3 into English word outputs

$$h_t = \phi(h_{t-1}, x_t) = f(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$$

$$h_t = \phi(h_{t-1}) = f(W^{(hh)}h_{t-1})$$

$$y_t = softmax(W^{(S)}h_t)$$

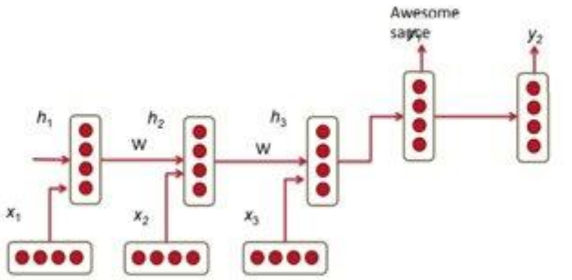
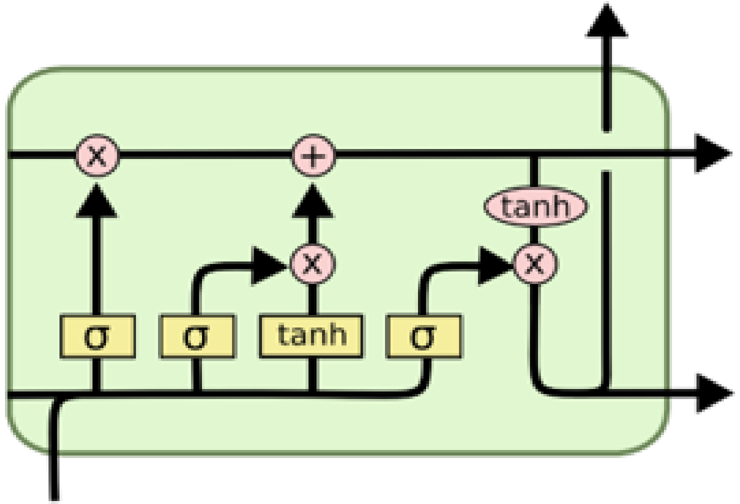


Figure 8: A RNN-based translation model. The first three RNN hidden layers belong to the source language model encoder, and the last two belong to the destination language model decoder.

4. LSTM

- How to capture long-term dependencies?

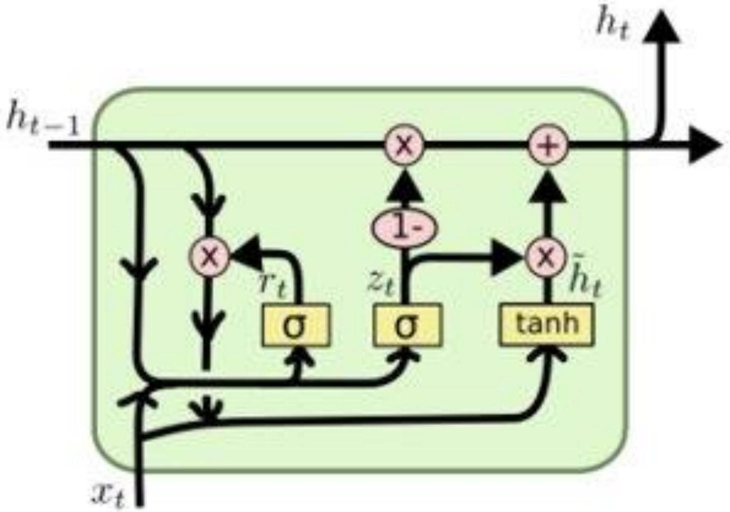
$$
\begin{aligned}
i_t &= \sigma(W^{(i)}x_t + U^{(i)}h_{t-1}) && \text{(Input gate)}\\
f_t &= \sigma(W^{(f)}x_t + U^{(f)}h_{t-1}) && \text{(Forget gate)}\\
o_t &= \sigma(W^{(o)}x_t + U^{(o)}h_{t-1}) && \text{(Output/Exposure gate)}\\
\tilde{c}_t &= \tanh(W^{(c)}x_t + U^{(c)}h_{t-1}) && \text{(New memory cell)}\\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t && \text{(Final memory cell)}\\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$



5. GRU

- Capture long-term dependencies simply

$i_t = \sigma(W^{(i)}x_t + U^{(i)}h_{t-1})$	(Input gate)
$f_t = \sigma(W^{(f)}x_t + U^{(f)}h_{t-1})$	(Forget gate)
$o_t = \sigma(W^{(o)}x_t + U^{(o)}h_{t-1})$	(Output/Exposure gate)
$\tilde{c}_t = \tanh(W^{(c)}x_t + U^{(c)}h_{t-1})$	(New memory cell)
$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$	(Final memory cell)
$h_t = o_t \circ \tanh(c_t)$	



THANK YOU!

Any question