Natural Language Processing (NLP) and Large Language Models (LLMs)
Lecture 6-2: Modern RNNs

Chendi Wang (王晨笛)
chendi.wang@xmu.edu.cn

WISE @ XMU

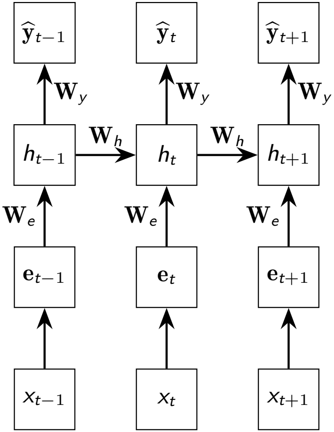
2025 年 4 月 8 日

Adapted in part from CS 224n and Dive into deep learning

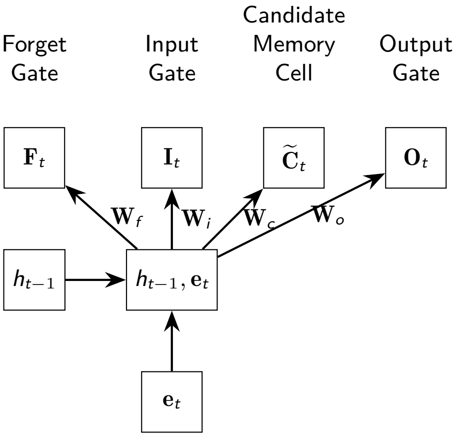
Guidance to your assignment

- A tutorial from our TA in bilibili
- A chinese blog
- May we extend it due to the mid-term?

Recap: vanilla RNN



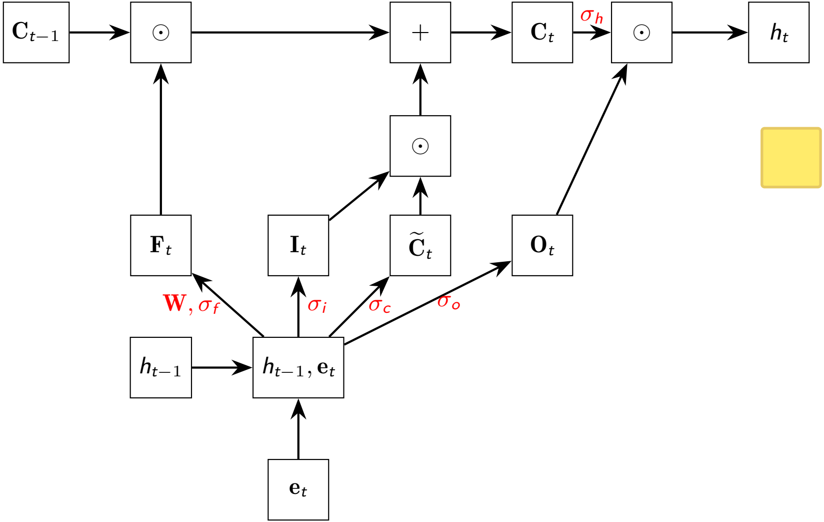
Forget gate, input gate, output gate



How to understand?

- Forget gate: whether to keep the current value.
- Input gate: how much of the input node's value should be added to the current memory cell
- Output gate:whether the memory cell should influence the output

Full graph


$\mathbf{C}_{t-1}$
$\odot$
$+$
$\mathbf{C}_t$
σ_h
$\odot$
h_t
$\odot$
$\mathbf{F}_t$
$\mathbf{I}_t$
$\widetilde{\mathbf{C}}_t$
$\mathbf{O}_t$
$\mathbf{W}, \sigma_f$
σ_i
σ_c
σ_o
h_{t-1}
$h_{t-1}, \mathbf{e}_t$
$\mathbf{e}_t$

Mathematical form of the gates

- $\mathbf{F}_t = \sigma_t(\mathbf{W}_{fh} h_{t-1} + \mathbf{W}_{fe}\mathbf{e}_t + \mathbf{b}_f)$
- $\mathbf{I}_t = \sigma_i(\mathbf{W}_{ih} h_{t-1} + \mathbf{W}_{ie}\mathbf{e}_t + \mathbf{b}_i)$
- $\mathbf{O}_t = \sigma_o(\mathbf{W}_{oh} h_{t-1} + \mathbf{W}_{oe}\mathbf{e}_t + \mathbf{b}_o)$
- $\sigma_f, \sigma_i, \sigma_o$ are usually sigmoid functions ($\sigma(t) = \frac{1}{1+\exp(-t)}$).

Mathematical form of the candidate memory cell

- $\widetilde{\mathbf{C}_t} = \sigma_c(\mathbf{W}_{ch}h_{t-1} + \mathbf{W}_{ce}\mathbf{e}_t + \mathbf{b}_c)$
- $\sigma_c = \tanh.$

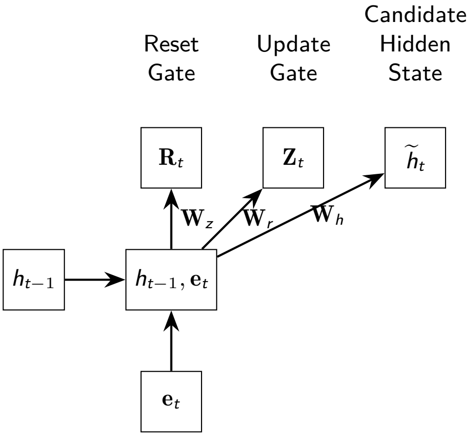
Mathematical form of the candidate memory cell

- $\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \widetilde{\mathbf{C}}_t$.
- Totally forget: $\mathbf{F}_t = [1, \cdots, 1]$ and $\mathbf{I}_t = [0, \cdots, 0]$

Output of the hidden layer

- $h_t = \sigma_h(\mathbf{C}_t) \odot \mathbf{O}_t$
- $\sigma_h = \tanh$

Graph


Reset Gate
Update Gate
Candidate Hidden State
$\mathbf{R}_t$
$\mathbf{Z}_t$
$\widetilde{h}_t$
$\mathbf{W}_z$
$\mathbf{W}_r$
$\mathbf{W}_h$
h_{t-1}
$h_{t-1}, \mathbf{e}_t$
$\mathbf{e}_t$

Math

- $\mathbf{R}_t = \sigma_t(\mathbf{W}_{rh} h_{t-1} + \mathbf{W}_{re}\mathbf{e}_t + \mathbf{b}_r)$
- $\mathbf{Z}_t = \sigma_z(\mathbf{W}_{zh} h_{t-1} + \mathbf{W}_{ze}\mathbf{e}_t + \mathbf{b}_z)$
- σ_t, σ_z are usually the sigmoid function.

Understand

- Reset gate: how much of the previous state we might still want to remember
- Update gate: how much of the new state is just a copy of the old one.

Candidate Hidden State

- $\widetilde{h}_t = \sigma_h \left(\mathbf{W}_{hh}(\mathbf{R}_t \odot h_{t-1}) + \mathbf{W}_{he}\mathbf{e}_t + \mathbf{b}_h\right)$
- $\mathbf{R}_t = [1, \cdots, 1]$: remember all previous results (vanilla RNN)
- $\mathbf{R}_t = [0, \cdots, 0]$: forget all previous results (MLP).
- $\sigma_h = \tanh$

Hidden state

- $h_t = \mathbf{Z}_t \odot h_{t-1} + (1 - \mathbf{Z}_t) \odot \widetilde{h}_t$
- $\mathbf{Z}_t = [1, \cdots, 1]$: no-update ($h_t = h_{t-1}$)

Deep RNNs

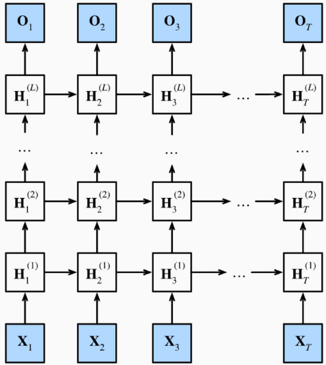


Fig. 10.3.1 Architecture of a deep RNN.

Figure is from `https://d2l.ai/chapter_recurrent-modern/deep-rnn.html`

Deep RNNs

- $$h_t^{[l]} = \sigma\left(\mathbf{W}_e^{[l]} h_t^{[l-1]} + \mathbf{W}_h^{[l]} h_{t-1}^{[l]} + \mathbf{b}_h^{[l]}\right)$$
- $$\mathbf{o}_t = \mathbf{W}_y h_t^{[L]} + \mathbf{b}_y$$
- $$\widehat{\mathbf{y}}^t = \mathrm{SoftMax}(\mathbf{o}_t).$$

Bidirectional task

- Next token prediction: The ______
- The ______ was coming down heavily, soaking the sidewalks.

Bidirectional RNN

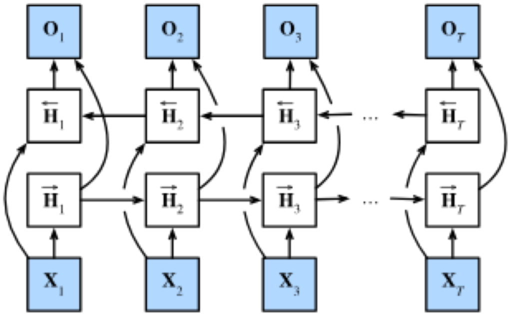


Figure is from `https://d2l.ai/chapter_recurrent-modern/bi-rnn.html`

Bidirectional RNN

- Forward:

$$\overrightarrow{h}_t = \sigma\left(\mathbf{W}_e \mathbf{e}_t + \mathbf{W}_h \overrightarrow{h}_{t-1} + \mathbf{b}_h^f\right)$$

- Backward:

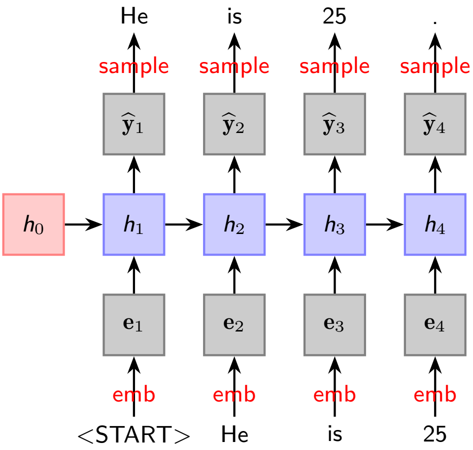
$$\overleftarrow{h}_t = \sigma\left(\mathbf{W}_e \mathbf{e}_t + \mathbf{W}_h \overleftarrow{h}_{t-1} + \mathbf{b}_h^b\right)$$

- $h_t = (\overrightarrow{h}_t, \overleftarrow{h}_t)$
- $\widehat{\mathbf{y}}^t = \mathrm{SoftMax}(\mathbf{W}_y h_t + \mathbf{b}_y)$.

Bidirectional model

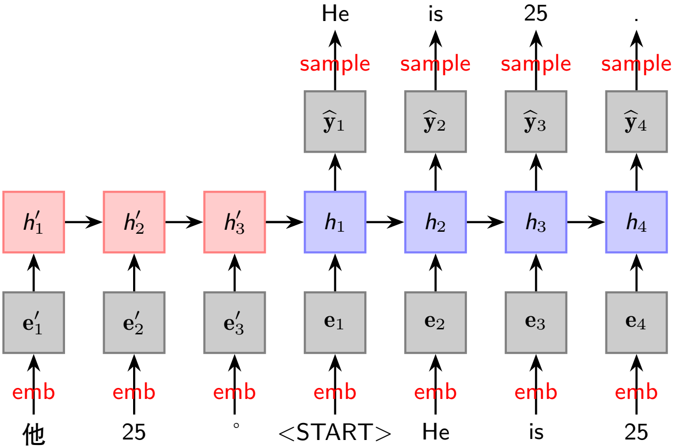
- Language model is not bidirectional as you have only access to previous tokens.
- If you have access to the whole sequence (such as translation, text analysis), , bidirectionality should be your first choice.
- BERT (Bidirectional Encoder Representations from Transformers) is a powerful bidirectional model.

The Encoder–Decoder Architecture



- h_0 can be an encoder RNN.

The Encoder–Decoder Architecture



The Encoder–Decoder Architecture

- The red part is called an encoder and the input sentence “他 25。” is called a source sentence.
- The blue part is called an decoder and the output sentence “He is 25.” is called a target sentence.
- The whole model is called a sequence-to-sequence (Seq2seq) model
- Applications:
  - Neural machine translation
  - Summarization (long text to short text); Dialogue; parsing (input sentence, output parse as sequence); Code generation

Statistical machine translation (1990s-2010s)

- Learn a probabilistic model from data for translation tasks.
- Probabilistic model: $p(\mathbf{x}_{\text{Eng}}|\mathbf{x}_{\text{Fra}})$.
- $\mathbf{x}_{\text{Eng}}$ is an English sequence and $\mathbf{x}_{\text{Fra}}$ is a French sequence.

Statistical machine translation (1990s-2010s)

- Find the best English sequence that maximizes the conditional probability :

$$\mathbf{x}_{\text{out}} = \text{Argmax}_{\mathbf{x}_{\text{Eng}}} \quad p(\mathbf{x}_{\text{Eng}} | \mathbf{x}_{\text{Fra}}).$$

- According to the Bayes rule, we have

$$\text{Argmax}_{\mathbf{x}_{\text{Eng}}} \quad p(\mathbf{x}_{\text{Eng}} | \mathbf{x}_{\text{Fra}}) = \text{Argmax}_{\mathbf{x}_{\text{Eng}}} \quad p(\mathbf{x}_{\text{Fra}} | \mathbf{x}_{\text{Eng}}) \cdot p(\mathbf{x}_{\text{Eng}}).$$

- $p(\mathbf{x}_{\text{Fra}} | \mathbf{x}_{\text{Eng}})$ is called a Translation Model that can be learned from parallel data. $p(\mathbf{x}_{\text{Eng}})$ is a Language Model learned from monolingual data.

Statistical machine translation (1990s-2010s)

- A statistical machine translation system is usually complicated.
- Feature design to capture particular language phenomenon.
- Maintain large corpus, including a large table of parallel data (such as equivalent phrases).
- For every pair of language, maintenance is reuqired.
- A lot of human effort!

Neural Machine Translation (NMT)

- Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. Cho et al., 2014.
- Sequence to Sequence Learning with Neural Networks. Sutskever et al., 2014.
- Google Translate switches from SMT to NMT since 2016.
- Another decisive win for neural network approaches!

Seq2seq models for machine translation

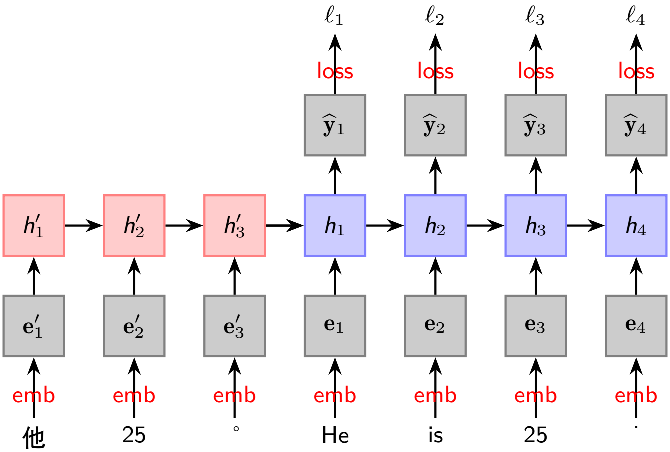
- Recap the conditional probability model $p(\mathbf{x}|\mathbf{x}')$.
- $\mathbf{x} = (x_1, \cdots, x_L)$ can be a English sequence and $\mathbf{x}' = (x'_1, \cdots, x'_{L_0})$ can be a French sentence.
- This is a conditional language model:

$$p(\mathbf{x}|\mathbf{x}') = p(x_1|\mathbf{x}') \prod_{i=2}^{L} p(x_i|x_1, \cdots, x_{i-1}, \mathbf{x}')$$

- Using the encoder-decoder RNN:

$$p(x_i|x_1, \cdots, x_{i-1}, \mathbf{x}') \approx p_\theta(x_i, h_{i-1}, h'_{L_0})$$

Train an Encoder–decoder Model



Train an Encoder-decoder model

- $h_t = \sigma\left(\mathbf{W}_h h_{t-1} + \mathbf{W}_e \mathbf{e}_t + \mathbf{b}_h\right)$
- $h_0 = h'_{L_0}$
- $h'_t = \sigma\left(\mathbf{W}'_h h'_{t-1} + \mathbf{W}_e \mathbf{e}'_t + \mathbf{b}'_h\right)$
- Loss function

$$\frac{1}{L}\sum_{t=1}^{L} \ell_t(\theta) = \frac{1}{L}\sum_{t=1}^{L} \mathrm{CE}(\mathbf{y}_t, \widehat{\mathbf{y}}_t)$$

- Train the weights using back-propagation.

Multi-layer

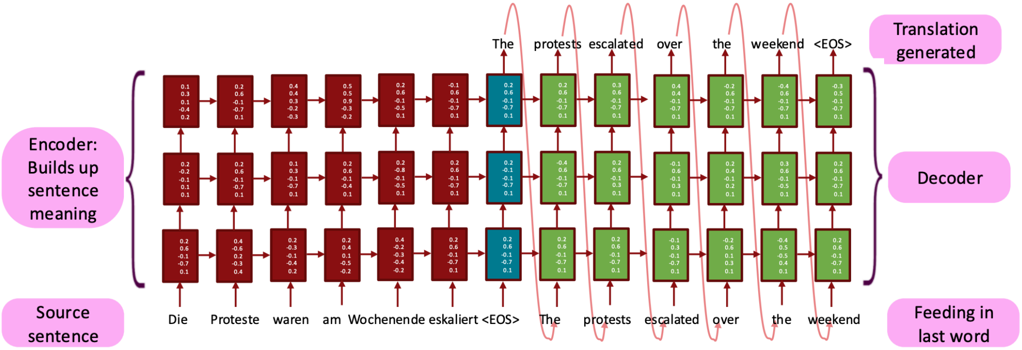


Figure is from CS 224n

Assignment 4 (TBD)

- Achieve a translation task using RNN.
- Here is a detailed guidance (RNN_NMT.ipynb).
- But your might be required to translate Chinese to English or English to Chinese.

Process your dataset

- Download and extract dataset (see our group).
- Pre-process the dataset.
- Tokenize the dataset to source and target. (source[i] is the i-th sequence)
- Plot the number of tokens for each sequence.

Construct a vocabulary

- Generate a vocabulary from the downloaded corpus, each token has an index based on its frequency.
- "<unk>" represents tokens that are not seen in the corpus.
- The start token "<bos>" and the end token "<eos>" are added.

Truncation and padding for simplicity

- The length for different sentences should be the same (controlled by "num_steps") for .
- If one sentence is too long, we truncate it.
- If one sentence is too short, we add a special token "<pad>" at the end of the sentence until it reaches num_steps.
- Add the "<eos>" token to the end of all sentences.

Read data

- Using *load_data_nmt*, we can obtain data iterator, source vocabulary, and target vocabulary
- An example of a small batch of dataset, X is the collection of source sequences and Y represents the target sequences.

Applying the encoder-decoder architecture

- Define an Encoder interface and the implementation will be provided by any model that inherits this base *Encoder* class.
- In the *Decoder* interface, add an additional *init_state* method to convert the encoder output into the encoded state.
- Decoders generate sequences step-by-step, using previous tokens and encoded states to predict the next token.
- The *EncoderDecoder* class is to concatenate the encoder and decoder.

Applying the Seq2seq model

- *Seq2SeqEncoder*: Structure of the RNN, making the size matchs RNN (X.permute)
- We apply a two-layer GRU with number of hidden units 16.
- *Seq2SeqDecoder*: repeat the hidden state of the encoder to make it matches the length of input sequence of the decoder.

Mask

- An example in the demo with

$$\text{mask} = [T, F, F \\ T, T, F]$$

- Masked Softmax CE loss (mask the added tokens “<pad>”)

Train

- Train using Adam

Evaluation

- We now have two sequences, the "True" sequence and the predicted sequence "Pred".
- Length penalty:

$$\exp\left(\min\left\{0, 1 - \frac{\text{len}_{\text{True}}}{\text{len}_{\text{Pred}}}\right\}\right)$$

- n-gram accuracy:

$$p_n = \frac{\#\ n\text{-grams in the pred sentence that matches the true one}}{\text{Total}\ \#\ \text{of}\ n\text{-grams in the pred sentence}}.$$

- An example True = [A, B, C, D, E, F], Pred = [A,B,B,C,D].
- Length penalty $= \exp(1 - 6/5)$; $p_1 = 4/5$ ($\#\{$A,B,C,D$\}/\#\{$A,B,B,C,D$\}$), $p_2 = 3/4$, $p_3 = 1/3$, $p_4 = 0$.

Evaluation

- We use the BLEU (bilingual evaluation understudy) score to evaluate the predicted sentence.
- BLEU score:

$$\text{Length penalty} \times p_n^{1/2^n}.$$

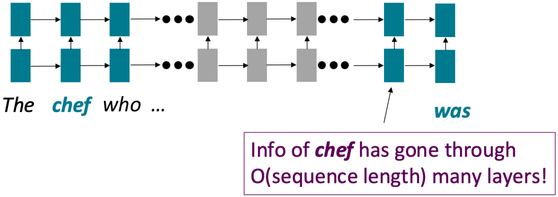
- True = Pred when BLEU = 1.

What is the difficulty of applying the encoder-decoder model to Chinese?

- Tokenization!

Issues with Rnns

- Linear interaction distance.



- Lack of parallelizability!

What's next?

- Attention is all you need.