165B
Machine Learning
Recurrent Neural Network

Lei Li (leili@cs)
UCSB

Outline

- Language Modeling
- Recurrent Neural Network
- Long-short term memory network (LSTM)
- Gated Recurrent Unit (GRU)
- Attention
- Encoder-decoder framework
- LSTM Seq2seq

Language Modeling

- Given a sentence y, estimate the probability

$$P(y) = \prod_t P(y_{t+1} \mid y_1 \cdots y_t)$$

$$P(y_{t+1} \mid y_1 \cdots y_t) = f_\theta(y_1, \ldots, y_t)$$

The cat sits on a __

y_1 y_2 y_3 y_4 y_5 y_6

$p(y_6 \mid y_1, \ldots, y_5)$

mat	0.15
rug	0.13
chair	0.08
hat	0.05
dog	0.01

Vocabulary

- To model P(y|x)
- Consider a ten-word sentence, chosen from common English dictionary about 5k words
  - 5000^{10} possible sentences
  - need a table of $5000^{10} \cdot 5000^{10}$ entries, infeasible
- source and target sentences need to break into smaller units.
- Multiple ways to segment
- Language specific considerations

Tokenization

- Break sentences into tokens, basic elements of processing
- Word-level Tokenization
  - Break by space and punctuation.
  - English, French, German, Spanish

  - Special treatment: numbers replaced by special token [number]
  - How large is the Vocabulary? Cut-off by frequency, the rest replaced by [UNK]

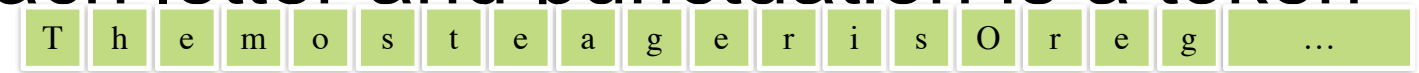
Pros and Cons of Word-level Tokenization

- Easy to implement
- Cons:
  - Out-of-vocabulary (OOV) or unknown tokens, e.g. Covid
  - Tradeoff between parameters size and unknown chances.
    - Smaller vocab => fewer parameters to learn, easier to generate (deciding one word from smaller dictionary), more OOV
    - Larger vocab => more parameters to learn, harder to generate, less OOV
  - Hard for certain languages with continuous script: Japanese, Chinese, Korean, Khmer, etc. Need separate word segmentation tool (can be neural networks)

最热切的是俄勒冈州，该州正在招募 5,000 名司机参与该国最大的试验。

Character-level Tokenization

- Each letter and punctuation is a token



- Pros:
  - Very small vocabulary (except for some languages, e.g. Chinese)
  - No Out-of-Vocabulary token
- Cons:
  - A sentence can be longer sequence
  - Tokens do not representing semantic meaning

Subword-level Tokenization

- Goal
  - The most eager is Oregon which is en listing 5,000 driver s in the country 's big g est experiment.
  - moderate size vocabulary
  - no OOV
- Idea:
  - represent rare words (OOV) by sequence of subwords
- Byte Pair Encoding (BPE)
  - not necessarily semantic meaningful
  - Originally for data compression

Philip Gage. A New Algorithm for Data Compression, 1994

Byte Pair Encoding

- Use smallest sequence of strings to represent original string. Group frequent pair of bytes together.
- Put all characters into symbol table
- For each loop, until table reach size limit
  - count frequencies of symbol pair
  - replace most frequent pair with a new symbol, add to symbol table

Byte Pair Encoding (BPE) for Text Tokenization

1. Initialize vocabulary with all characters as tokens (also add end-of-word symbol) and frequencies
2. Loop until vocabulary size reaches capacity
    1. Count successive pairs of tokens in corpus
    2. Rank and select the top frequent pair
    3. Combine the pair to form a new token, add to vocabulary
3. Output final vocabulary and tokenized corpus

Rico Sennrich et al. Neural Machine Translation of Rare Words with Subword Units. 2016

Example

l, o, w, e, r, n, s, t, i, d, </w>	‘l o w</w>’ : 5	‘l o w e r</w>’ : 2 ’n e w e s t</w>’ : 6 ‘w i d e s t</w>’ : 3
l, o, w, e, r, n, s, t, i, d, </w>, es	‘l o w</w>’ : 5	‘l o w e r</w>’ : 2 ’n e w es t</w>’ : 6 ‘w i d es t</w>’ : 3
l, o, w, e, r, n, s, t, i, d, </w>, es, est	‘l o w</w>’ : 5	‘l o w e r</w>’ : 2 ’n e w est</w>’ : 6 ‘w i d est</w>’ : 3
l, o, w, e, r, n, s, t, i, d, </w>, es, est, est</w>	‘l o w</w>’ : 5	‘l o w e r</w>’ : 2 ’n e w est</w>’ : 6 ‘w i d est</w>’ : 3
l, o, w, e, r, n, s, t, i, d, </w>, es, est, est</w>, lo,	‘lo w</w>’ : 5	‘lo w e r</w>’ : 2 ’n e w est</w>’ : 6 ‘w i d est</w>’ : 3
l, o, w, e, r, n, s, t, i, d, </w>, es, est, est</w>, lo, low	‘low</w>’ : 5	‘low e r</w>’ : 2 ’n e w est</w>’ : 6 ‘w i d est</w>’ : 3

Predict Next Token Probability

There are many methods to predict the next token:

- N-gram: assuming

$$p\left(x_t \middle| x_1, \ldots, x_{t-1}\right) = p\left(x_t \middle| x_{t-k}, \ldots, x_{t-1}\right)$$

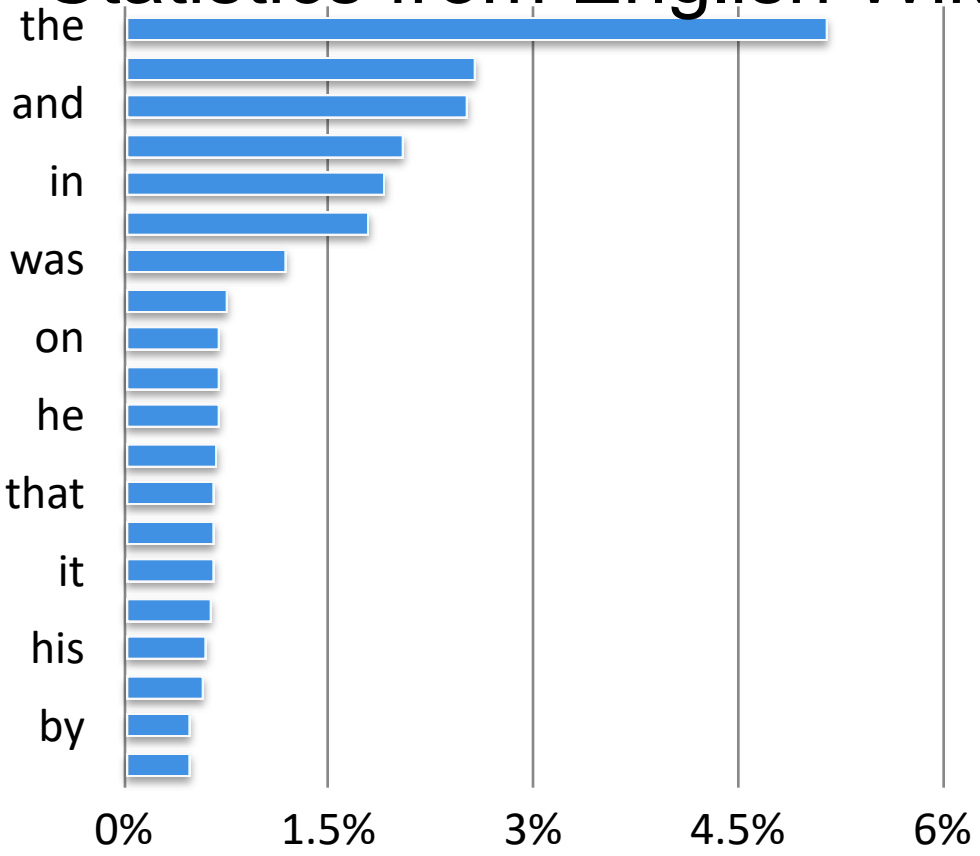
  , and estimate it directly
- Context MLP: use DNN to estimate

$$p\left(x_t \middle| x_{t-k}, \ldots, x_{t-1}\right)$$

- CNN-LM (previous lecture)
- RNN-LM, LSTM, GRU
- GPT

Word and Bigram

Statistics from English Wikipedia and books



cond. prob. $p(x_2|x_1)$

	first	united	the	a	be
the	0.014	0.006			
of			0.283	0.030	
would					0.191
with			0.187	0.122	

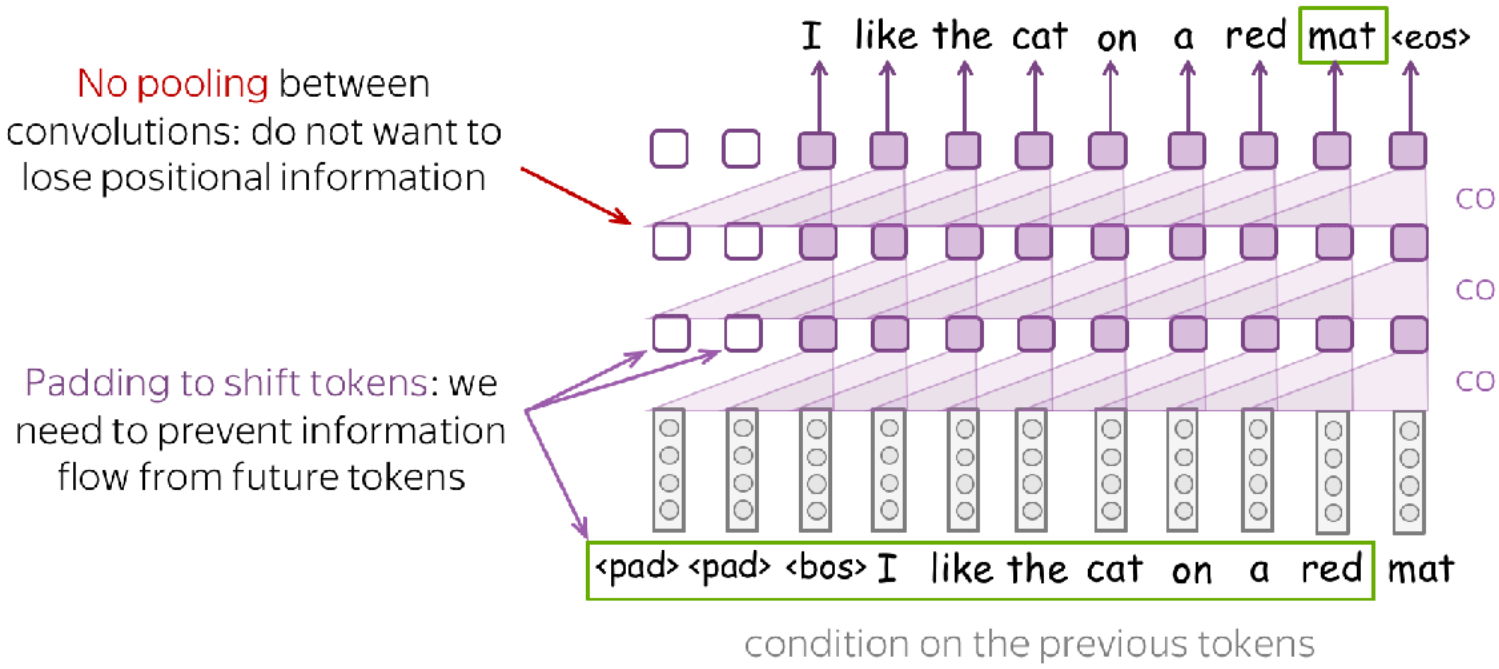
Challenge of n-gram LM

- Vocabulary: V
- n-gram needs a probability table of size V^n
- Common V size 30k ~ 100k
- Hard to estimate and hard to generalize
- Solution: Parameterization with generative model
  - $p(y_t | y_1, \cdots, y_{t-1}; \theta) = f_\theta(y_1, \cdots, y_{t-1})$
  - f can be a carefully designed and computationally tractable function, e.g. a neural network (later lectures).

CNN Language Model

$$P(y_{t+1} \mid y_1, \ldots, y_t) \approx \mathrm{CNN}_\theta(y_{t-k}, \ldots, y_t)$$

predict the next token



But,

limited cont

Stack several convolutions

https://lena-voita.github.io/nlp_course/models/convolutional.html

Limitation of CNN-LM

- CNN-LM only has a fixed-length receptive field
  - probability of next token only dependent on a fixed-size context
- But sentences are of variable length
- How to handle sentences with variable length?
- Idea:
  - adding memory to network
  - adaptive updating memory

Recurrent Memory

- Introduce memory representation
- RNN-LM: use RNN to estimate

$$p\left(x_t \middle| x_1, \ldots, x_{t-1}\right) = \text{softmax}(W \cdot h_t)$$

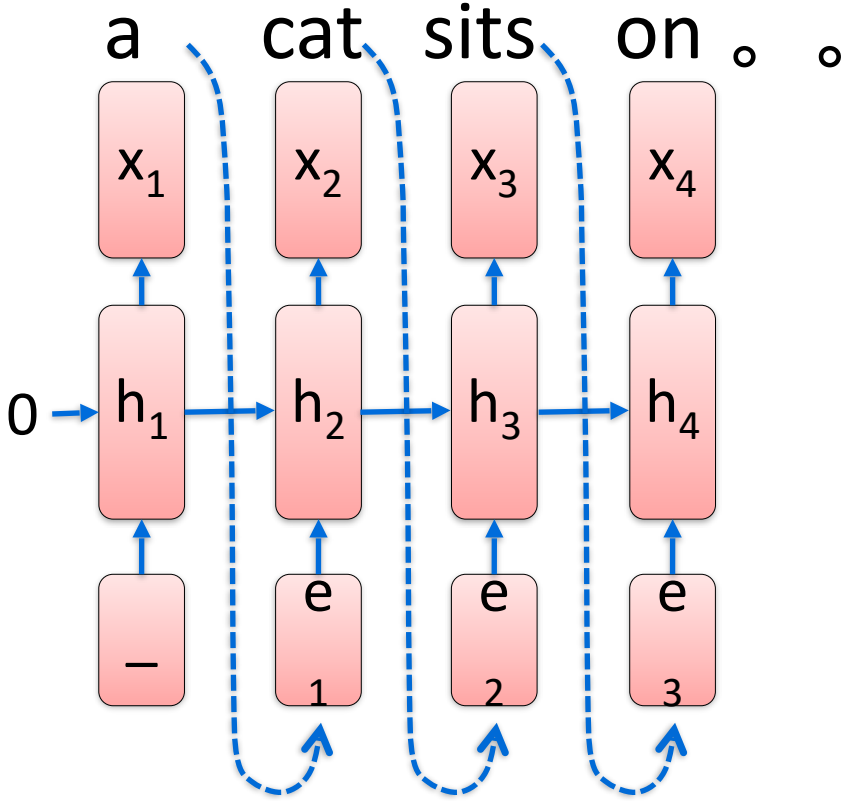
$$h_t = RNN(h_{t-1}, Emb(x_{t-1}))$$

- RNN cell can be
  - Simple feedforward neural network
  - Long-short term memory
  - Gated recurrent units

Recurrent Neural Network

$$p\left(x_t \middle| x_1, \ldots, x_{t-1}\right) = \text{softmax}\left(U \cdot h_t\right)$$

$$h_t = \sigma\left(W \cdot \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b\right)$$



Elman, Finding Structure in Time. Cog. Sci. 1990.

Mikolov et al, Recurrent neural network based language model. Interspeech 2010.

Training RNN-LM

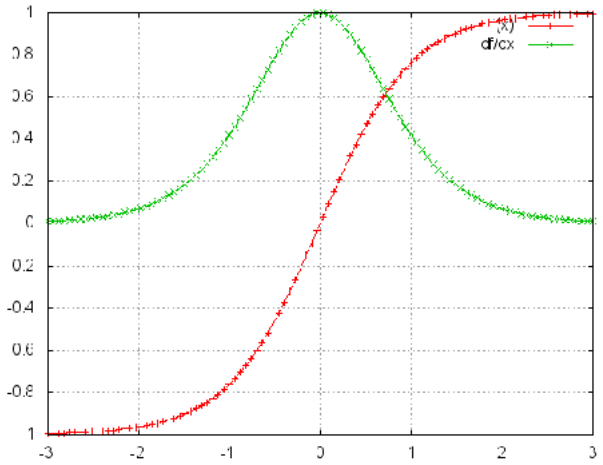
- Risk:
  - Loss: cross-entropy for every next-token given prefix context
  - CE(x_t+1, f(x_1, …, x_t))
- SGD
  - Calculate gradient: Back-propogation through time (BPTT)
  - ∇E_t

Back-propagation for RNN (python)

```python
def bptt(self, x, y):
    T = len(y)
    # Perform forward propagation
    o, s = self.forward_propagation(x)
    # We accumulate the gradients in these variables
    dLdU = np.zeros(self.U.shape)
    dLdV = np.zeros(self.V.shape)
    dLdW = np.zeros(self.W.shape)
    delta_o = o
    delta_o[np.arange(len(y)), y] -= 1.
    # For each output backwards...
    for t in np.arange(T)[::-1]:
        dLdV += np.outer(delta_o[t], s[t].T)
        # Initial delta calculation: dL/dz
        delta_t = self.V.T.dot(delta_o[t]) * (1 - (s[t] ** 2))
        # Backpropagation through time (for at most self.bptt_truncate steps)
        for bptt_step in np.arange(max(0, t-self.bptt_truncate), t+1)[::-1]:
            # Add to gradients at each previous step
            dLdW += np.outer(delta_t, s[bptt_step-1])
            dLdU[:,x[bptt_step]] += delta_t
            # Update delta for next step dL/dz at t-1
            delta_t = self.W.T.dot(delta_t) * (1 - s[bptt_step-1] ** 2)
    return [dLdU, dLdV, dLdW]
```

Computational Issue: Gradient Vanishing

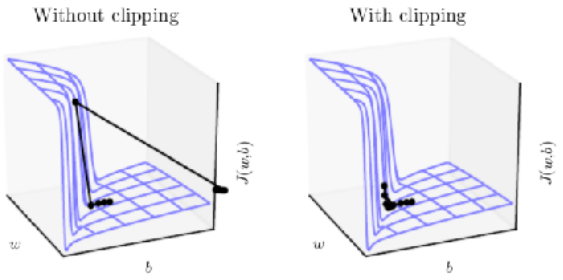
- tanh has derivative close to zero at both ends



Pascanu et al. On the difficulty of training recurrent neural networks. ICML 2013

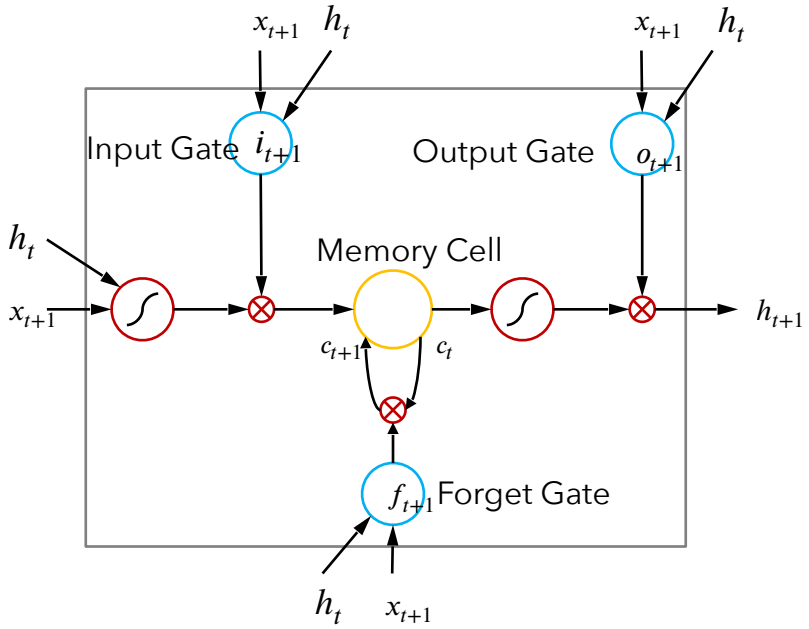
Gradient Exploding

- Use gradient clipping
- Two options: clip by absolute value or rescale norm
- if $|g| > \eta, \hat{g} \leftarrow \eta$
- if $|g| > \eta, \hat{g} \leftarrow \frac{\eta}{|g|} g$



Long-Short Term Memory (LSTM)

- Replace cell with more advanced one
- Adaptively memorize short and long term information



$$i_{t+1} = \sigma(M_{ix}x_{t+1} + M_{ih}h_t + b_i)$$
$$f_{t+1} = \sigma(M_{fx}x_{t+1} + M_{fh}h_t + b_f)$$
$$o_{t+1} = \sigma(M_{ox}x_{t+1} + M_{oh}h_t + b_o)$$

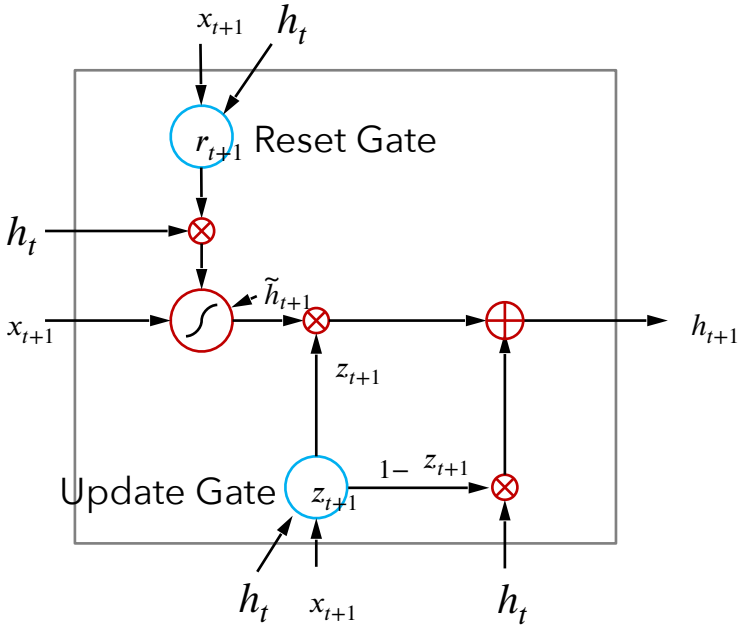
$$a_{t+1} = \tanh(M_{cx}x_{t+1} + M_{ch}h_t + b_a)$$

$$c_{t+1} = f_{t+1} \otimes c_t + i_{t+1} \otimes a_{t+1}$$
$$h_{t+1} = o_{t+1} \otimes \tanh(c_{t+1})$$

Hochreiter & Schmidhuber. Long Short-Term Memory, 1997
Gers et al. Learning to Forget: Continual Prediction with LSTM. 2000

Gated Recurrent Unit (GRU)

- Adaptively memorize short and long term information
- like LSTM, but fewer parameters



Input: x_t

Memory: h_t

$$r_{t+1} = \sigma(M_{rx}x_{t+1} + M_{rh}h_t + b_r)$$
$$z_{t+1} = \sigma(M_{zx}x_{t+1} + M_{zh}h_t + b_z)$$

$$\tilde{h}_{t+1} = \tanh(M_{hx}x_{t+1} + M_{hh}(r_{t+1} \otimes h_t) + b_h)$$

$$h_{t+1} = z_{t+1} \otimes \tilde{h}_{t+1} + (1 - z_{t+1}) \otimes h_t$$

Cho et al. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. 2014

Next Up

- Attention
- Sequence Labelling
- Sequence-to-sequence model