香港城市大學（東莞）
City University of Hong Kong
(Dongguan)

Celebrating the Establishment in Year 2024

Lecture 2: Language Model

CS6493 Natural Language Processing

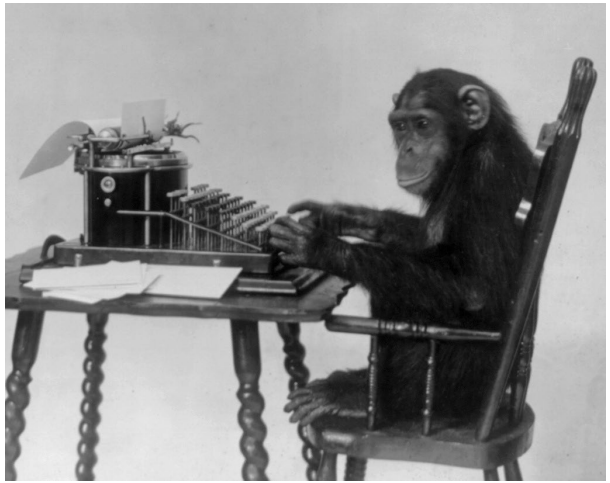
Instructor: Linqi Song

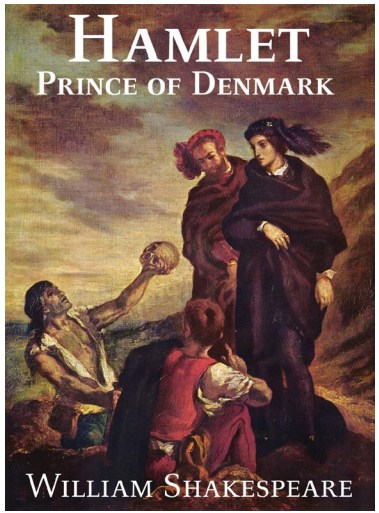
Outline

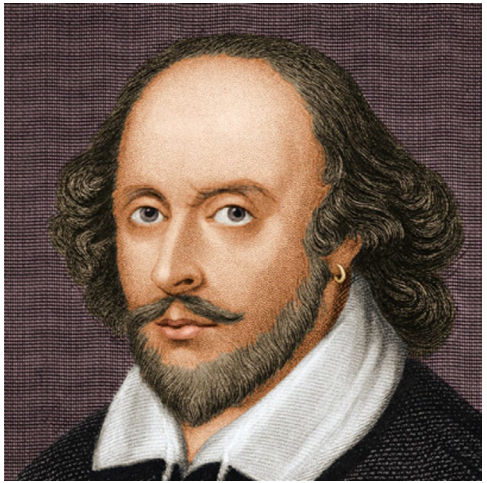
- 1. Language model definition & applications
- 2. Model construction
  - Statistical language models
  - Neural language models
- 3. Language model evaluation

The famous infinite monkey theorem

- The famous infinite monkey theorem
  - A monkey hitting keys at random on a typewriter keyboard for an infinite amount of time will almost surely type any given text, such as Hamlet or even the complete works of William Shakespeare.
  - How can we judge the performance of each typing?







What is a language model?

- A language model is a probability distribution over sequences of words. Given such a sequence, say of length *T*, it assigns a probability $P(x^{(1)}, x^{(2)}, \dots x^{(T)})$ to the whole sequence.

- To determine whether a given ordering of words sounds like natural language.

  E.g., two orderings
  'NLP is about how machines understand and process natural languages.'
  'machines NLP about languages understand how and is natural process.'

Another interpretation of a language model

- The task of language models could be regarded as assigning probability to a piece of text.
- For example, consider a piece of text $(x^{(1)}, x^{(2)}, \dots x^{(T)})$, then according to the Langugage Model, the probability of the text could be written as:

$$P(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \ \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \ \boldsymbol{x}^{(T-1)}, \dots, \boldsymbol{x}^{(1)})$$

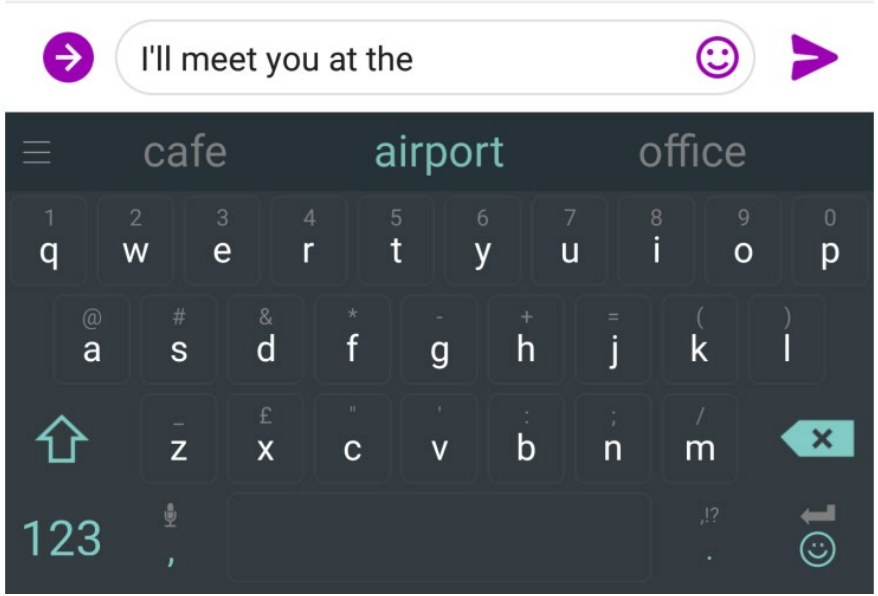
$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \ \boldsymbol{x}^{(t-1)}, \dots, \boldsymbol{x}^{(1)})$$

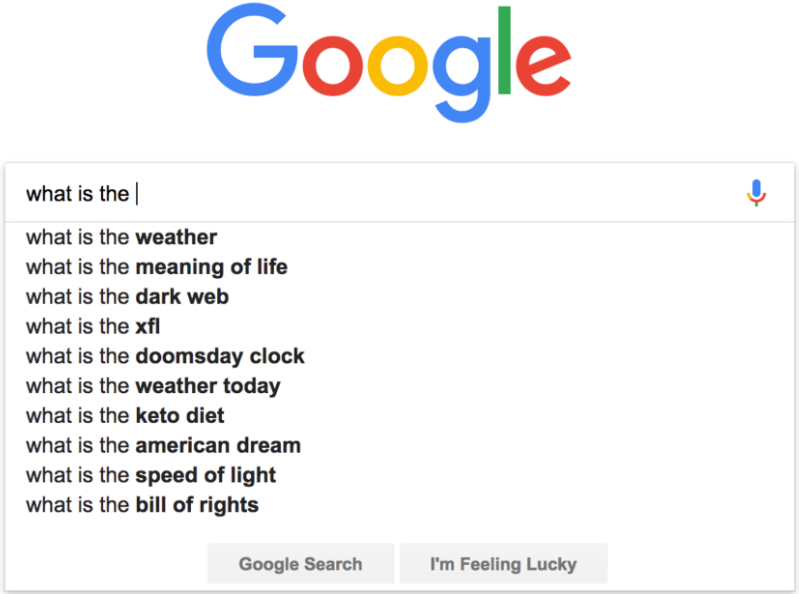
A language model can also be interpreted as predicting what word comes next

Why we care about language modeling?

- It serves as a benchmark task that helps us to measure our progress on understanding language
- It is a subcomponant of many NLP tasks, especially those involving generating text and estimating the probability of the text:
  - predictive typing
  - speech recognition
  - handwriting recognition
  - spelling / grammar correction
  - authorship identification
  - machine translation
  - summarization
  - dialogue...

Language models in our daily life (1)





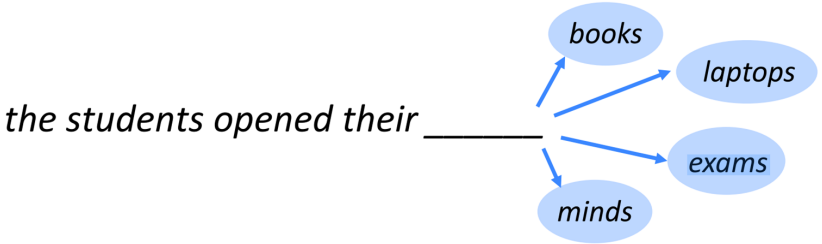
Language models in our daily life (2)

Speech recognition

I ate a cherry.	Eye eight uh Jerry.
Birds can fly in the sky.	Beards can fly in the sky.
Recognize speech.	Wretch a nice beach.

Language modeling

- The task to predict what word comes next.



- Given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$; compute the probability distribution of the next word $x^{(t+1)}$:

$$P\left(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(1)}\right)$$

where $x^{(t+1)}$ could be any word in the vocabulary $V = \left\{w_1, \ldots, w_{|V|}\right\}$

- Learning this language model is called Language Modeling.

n-gram model assumption

- **Markov assumption**: $x^{(t+1)}$ depends only on the preceding *n-1* words.

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}}) \quad \text{(assumption)}$$

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(definition of conditional prob)}$$

- Get these n-gram and (n-1)-gram probabilities by counting them in some large corpus of texts.

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(statistical approximation)}$$

n-gram model

- Every word in Σ is assigned some probability, conditoned on a fixed-length history (n – 1).
- An n-gram is a chunk of *n* consecutive words.

the students opened their _____

unigrams: “the”, “students”, “opened”, ”their”
bigrams: “the students”, “students opened”, “opened their”
trigrams: “the students opened”, “students opened their”
4-grams: “the students opened their”

Unigram model

- Every word in the vocabulary Σ is assigned some probability.
- Random variables $x^{(1)}, x^{(2)}, \dots x^{(T)}$ (one per word).

$$P\left(x^{(1)}, x^{(2)}, \dots x^{(T)}\right) = \Pi_{t=1}^{T} P\left(x^{(t)} | x^{(1)}, x^{(2)}, \dots, x^{(t-1)}\right) = \Pi_{t=1}^{T} P\left(x^{(t)}\right)$$

$x^{(0)} =< s >$, starting symbol

Defficiency of the model: hard to distinguish muiti-word phrases.

E.g.: "Artificial Intelligence is the electricity for the new era. "

Here 'Artificial' and 'Intelligence' are considered as two separate word, while the semantic linkage between them are much stronger than other adjacent word pairs in the sentence.

Unigram distribution examples

[rank 1]
p(the) = 0.038
p(of) = 0.023
p(and) = 0.021
p(to) = 0.017
p(is) = 0.013
p(a) = 0.012
p(in) = 0.012
p(for) = 0.009
…

…
[rank 1001]
p(joint) = 0.00014
p(relatively) = 0.00014
p(plot) = 0.00014
p(DEL1SUBSEQ) = 0.00014
p(rule) = 0.00014
p(62.0) = 0.00014
p(9.1) = 0.00014
p(evaluated) = 0.00014
…

Probability distribution of a unigram model

Generate texts by selecting the word with a certain probability as the next output word.

first, from less the This different 2004), out which goal 19.2 Model their It ~(i?1), given 0.62 these (x0; match 1 schedule. x 60 1998. under by Notice we of stated CFG 120 be 100 a location accuracy If models note 21.8 each 0 WP that the that Nov?ak. to function; to [0, to different values, model 65 cases. said - 24.94 sentences not that 2 In to clustering each K&M 100 Boldface X))] applied; In 104 S. grammar was (Section contrastive thesis, the machines table -5.66 trials: An the textual (family applications. We have for models 40.1 no 156 expected are neighborhood

Text generated from a unigram model

Full history model

- Every word in Σ is assigned some probability, conditioned on every history.

$$P\left(x^{(1)}, x^{(2)}, \dots x^{(T)}\right) = \Pi_{t=1}^{T} P\left(x^{(t)} | x^{(1)}, x^{(2)}, \dots, x^{(t-1)}\right)$$

A text generation example of a full history model:

Bill Clinton's unusually direct comment Wednesday on the possible role of race in the election was in keeping with the Clintons' bid to portray Obama, who is aiming to become the first black U.S. president, as the clear favorite, thereby lessening the potential fallout if Hillary Clinton does not win in South Carolina.

Problems with n-gram

Suppose we are learning a 4-gram Language Model.

students opened their _____

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

Sparsity Problem 1

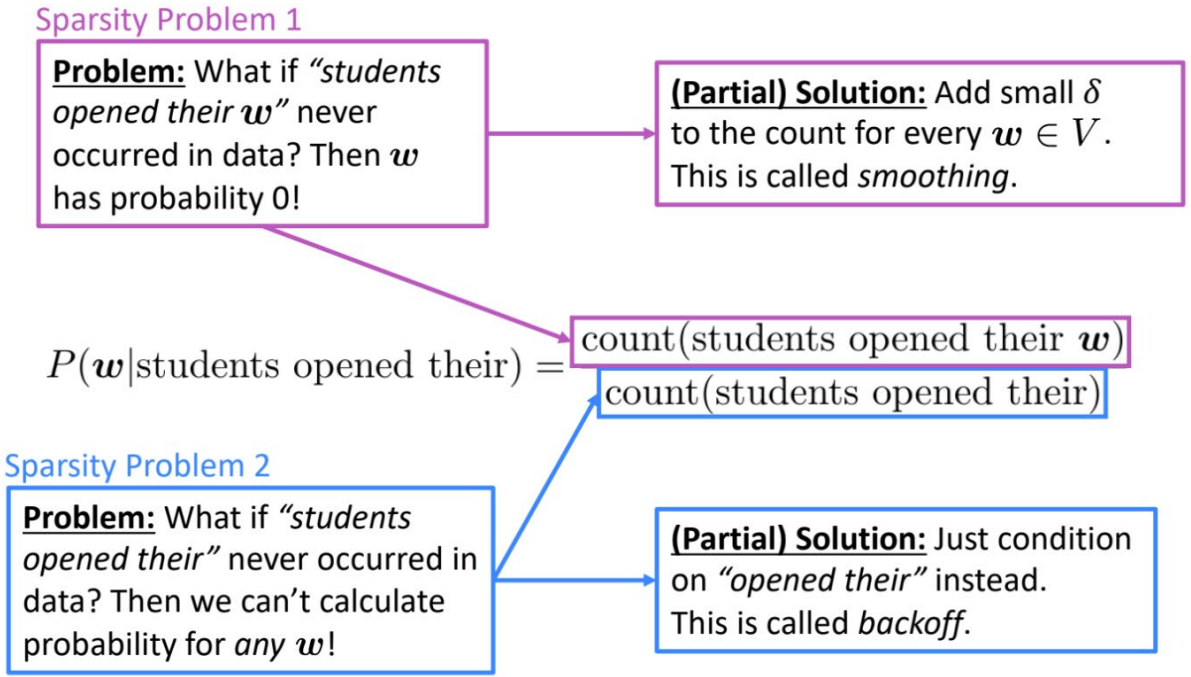
Problem: What if *"students opened their* $\boldsymbol{w}$*"* never occurred in data? Then $\boldsymbol{w}$ has probability 0!

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* $\boldsymbol{w}$!

Storage: Need to store count for all *n*-grams you saw in the corpus.

Sparsity problems with n-gram

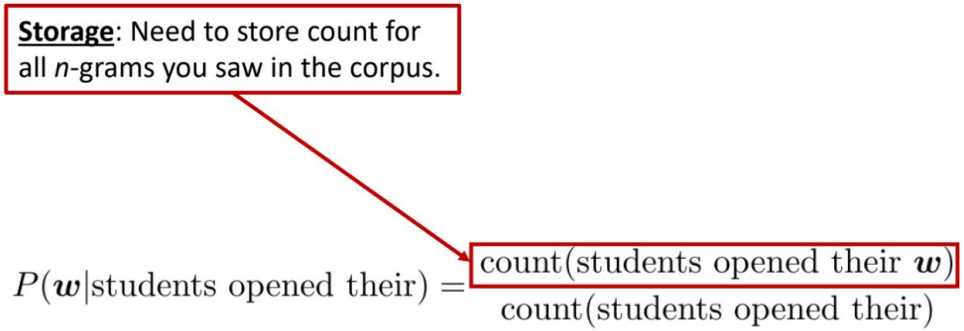


Note:

Increasing *n* makes sparsity problems worse.

Typically we can't have *n* bigger than 5.

Storge problems with n-gram



Note:

Increasing corpus increases the model size.

n-gram in text generation

- A passage generated with a 3-gram model:

  today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

- Well grammarly-structured.
- But incoherent, as each calculation only considers 3 continuous words.
- Yet increasing n brings sparsity and storage problems.
- Solution: window-based neural network.

A fixed-window neural language model

Y. Bengio, et al. (2003): A Neural Probabilistic Language Model

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

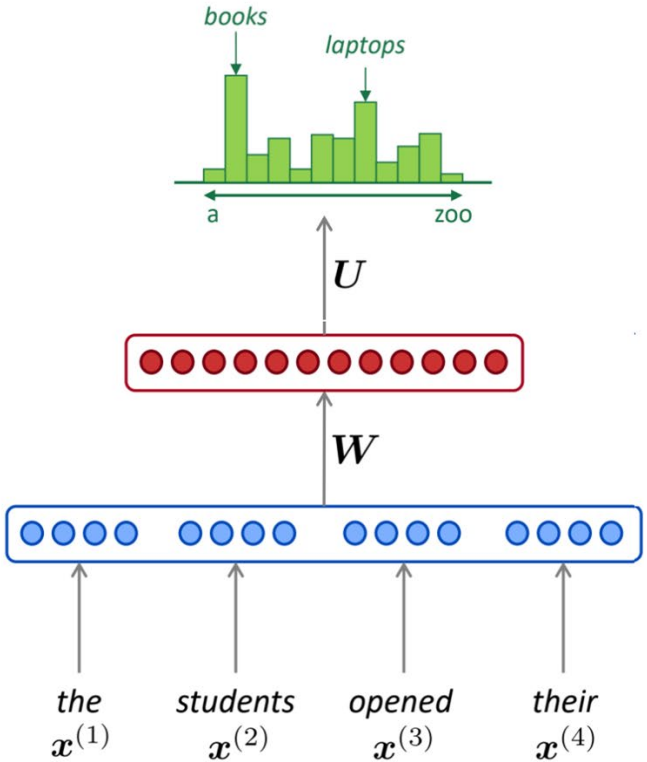
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



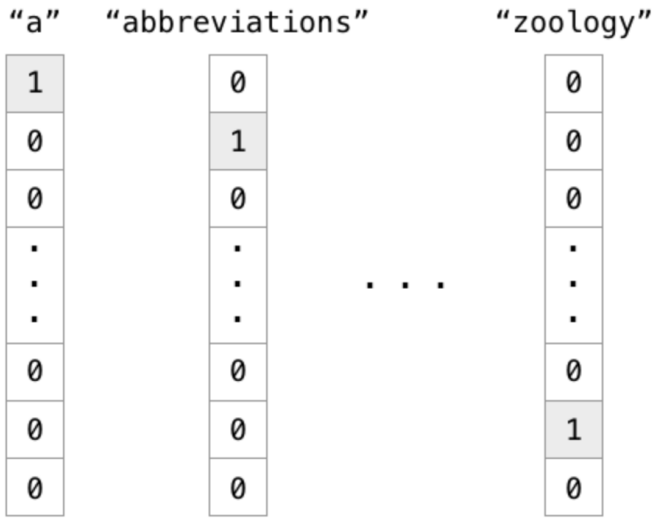
Improvements over n-gram:

- No sparsity problem
- Don't need to store all observed n-grams

Remaining problems:

- Fixed window is too small.
- Enlarging window enlarges $\boldsymbol{W}$; Window can never be large enough.
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in $\boldsymbol{W}$. No symmetry in how the inputs are processed.

One hot encoding and word embeddings



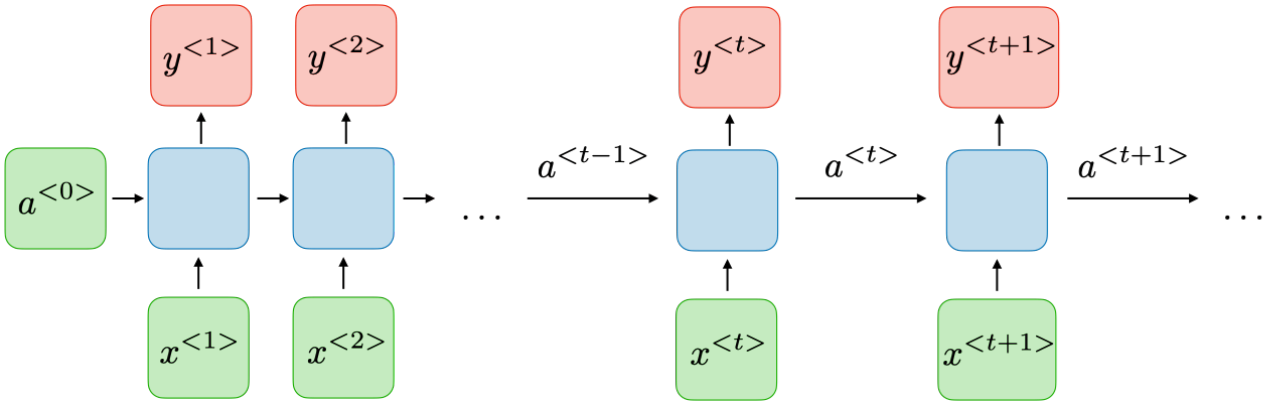
One hot encoding

$$\underbrace{\begin{bmatrix}0 & 0 & 0 & 1 & 0\end{bmatrix}}_{\text{One-hot vector}} \times \underbrace{\begin{bmatrix}8 & 2 & 1 & 9\\ 6 & 5 & 4 & 0\\ 7 & 1 & 6 & 2\\ 1 & 3 & 5 & 8\\ 0 & 4 & 9 & 1\end{bmatrix}}_{\text{Embedding Weight Matrix}} = \underbrace{\begin{bmatrix}1 & 3 & 5 & 8\end{bmatrix}}_{\text{Hidden layer output}}$$

Word embeddings

Recall of Recurrent Neural Networks (RNNs)

- Recurrent units: blocks share the structure and parameters (weights)
- Flexible input / output size
- Self-connections or connections to units in the previous layers
- Short-term memory

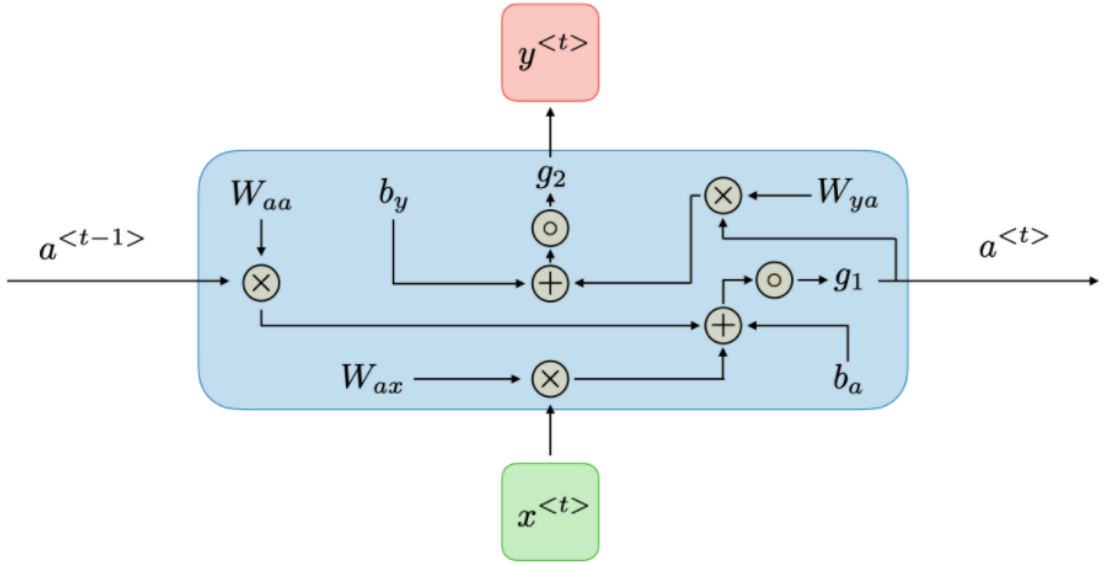


RNN block

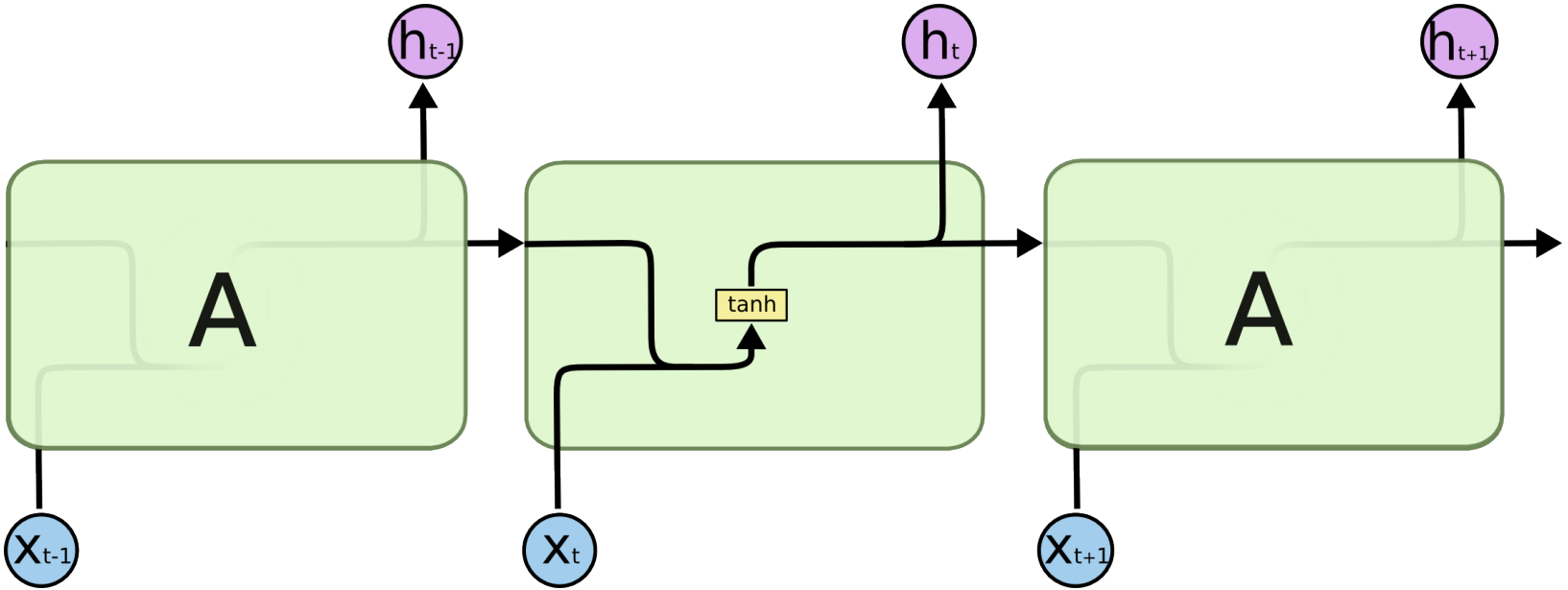
For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

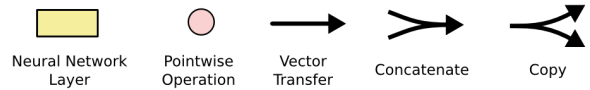
where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions.



RNN block concatenation

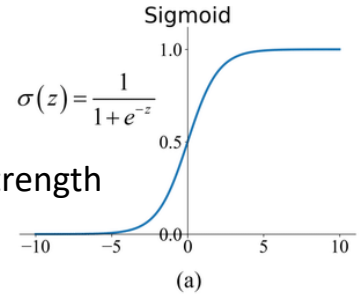


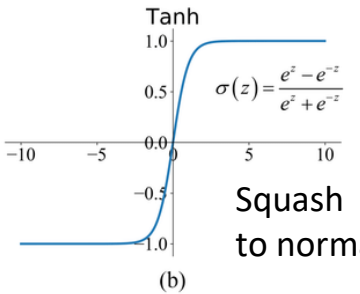
- Another form: usually use hidden state h_t



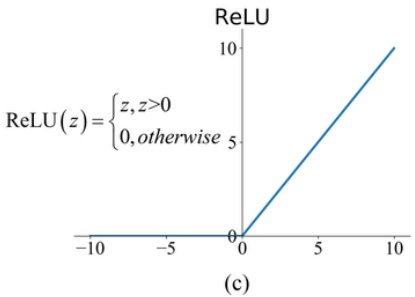
Activation functions

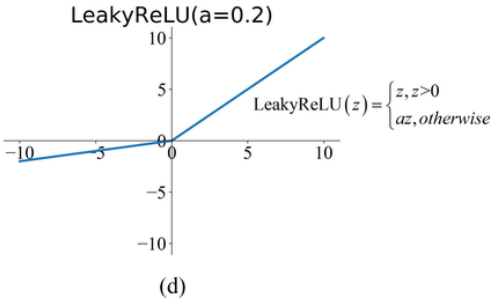
Soft switch
to control signal flow strength





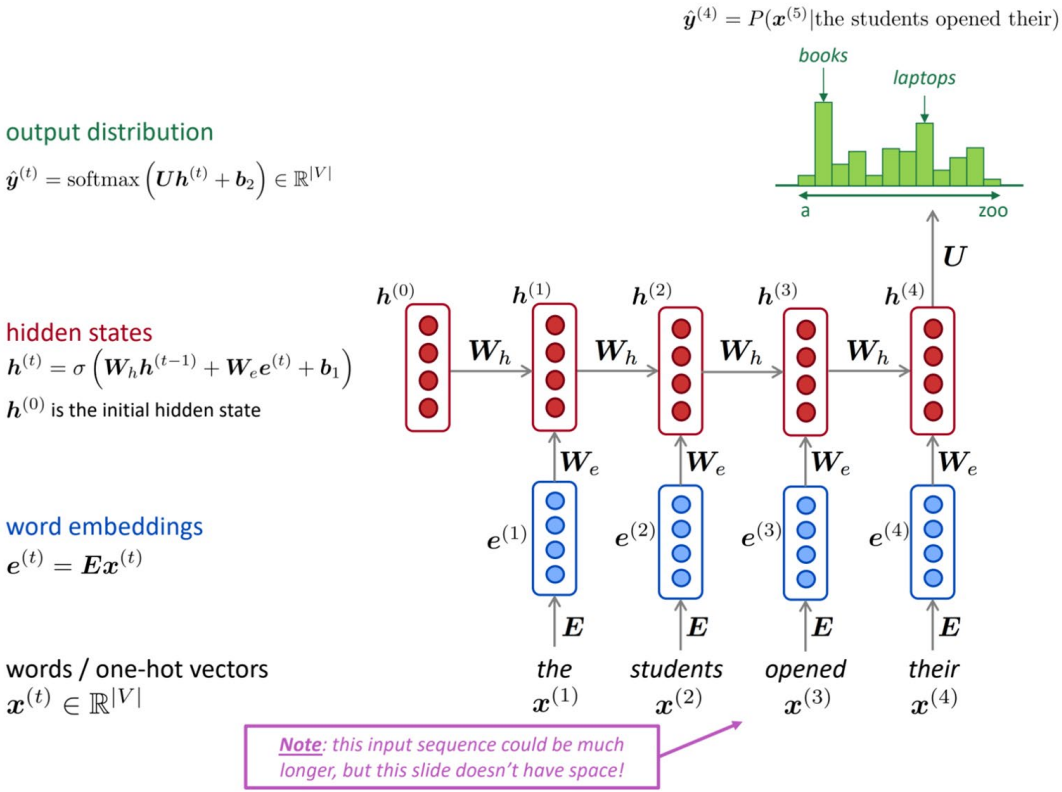
Squash
to normalize values to be between 0 and 1





Activation functions

RNN-based language model



RNN Advantages:

- Can process any length input
- Computation for step t can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

Training an RNN-LM

- Get a big corpus of text which is a sequence of words
- Feed into RNN-LM; compute output distribution for every step t.
  i.e., predict probability distribution of every word, given words so far
- Loss function on step *t*: cross-entropy between predicted probability distribution $\hat{y}^{(t)}$, and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$):

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

- Average the result above to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

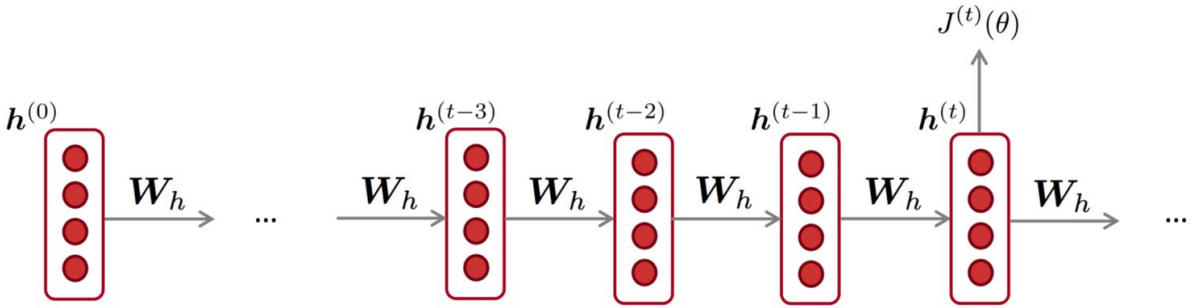
Training an RNN-LM

- Computing loss and gradients across entire corpus is too expensive.

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)$$

- In practice, consider $x^{(1)}, \ldots, x^{(T)}$ as a sentence (document).
- Stochastic Gradient Descent (SGD) allows us to compute loss and gradients for small chunks of data, and update.
- Compute loss for a sentence (actually a batch of sentences) using SGD, compute gradients and update weights. Repeat.

Backpropagation through time



$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)$$

- How to calculate the derivatives of $J(\theta)$ with respect to the repeated weight matrix W_h through time $J^{(t)}(\theta)$?
- Hints: multivariable chain rule.

Vanishing/Exploding gradient

Loss function of RNN

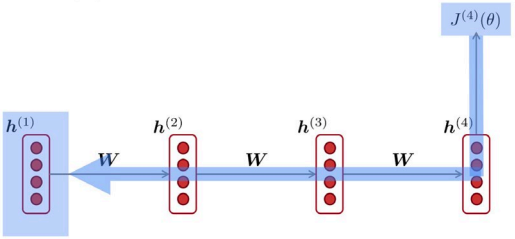
$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

In the case of a recurrent neural network, the loss function $\mathcal{L}$ of all time steps is defined based on the loss at every time step, e.g., cross entropy loss.

Backpropagation through time

$$\frac{\partial \mathcal{L}^{(T)}}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}^{(T)}}{\partial W}\Big|_{(t)}$$

Backpropagation is done at each point in time.
W: weight matrix
T: the T-th timestamp



$$\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(1)}} = \frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(T-1)}} \cdots \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}}.$$

When T is large enough, the result of the multiplied gradient could increase/decrease exponentially with respect to the number of layers, leading to ineffective weight updates.

$1.01^{365}=37.8$

$0.99^{365}=0.03$

Question: *What's the possible influence of gradient vanishing/exploding?*

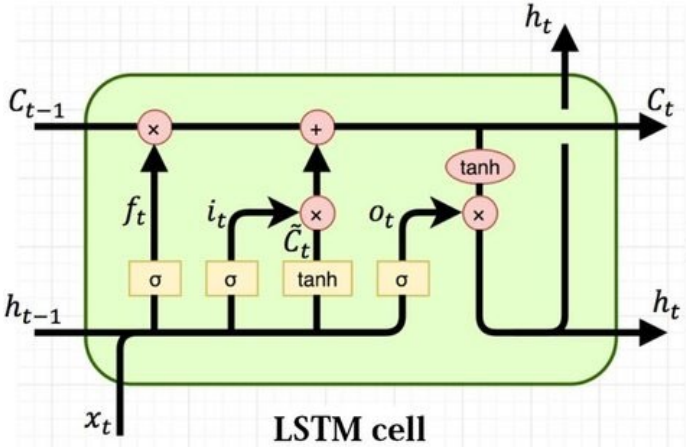
Long-term dependency in NLP

- **LM task**: When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____.

- To learn from this training example, the model needs to model the dependency between “tickets” on the 7th step and the target word “tickets” at the end.
- If gradient is small, the model can’t learn this dependency
- So the model is unable to predict similar long-distance dependencies at test time

Long Short-Term Memory (1)

- Long Short-Term Memory, a special RNN to slove vanish gradient phenomenon (Hochreiter and Schmidhuber, 1997).
- Instead of adding up the history, it uses a set of gating units to control the memory.



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$: input gate

$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$: forget gate

$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$: output gate

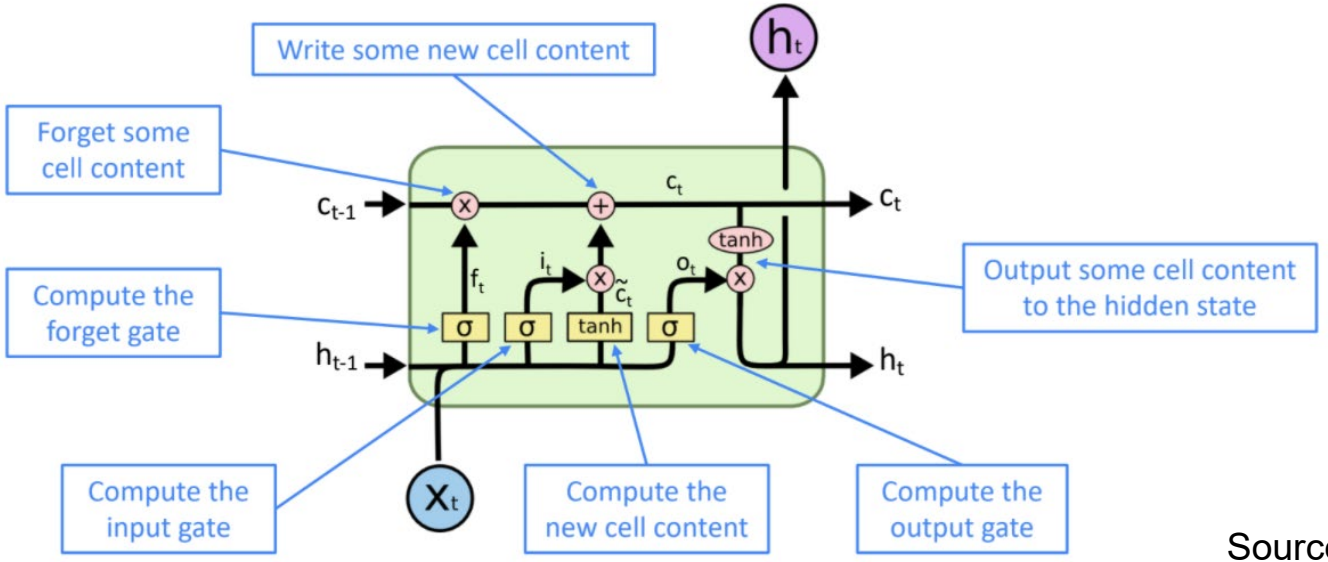
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$: new content of input t

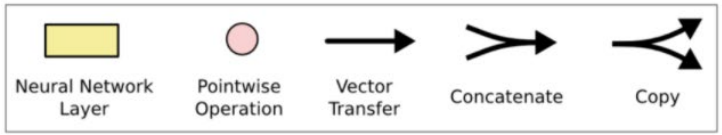
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$: cell state

$$h_t = \tanh(C_t) * o_t$$: hidden state

Long Short-Term Memory (2)

You can think of the LSTM equations visually like this:

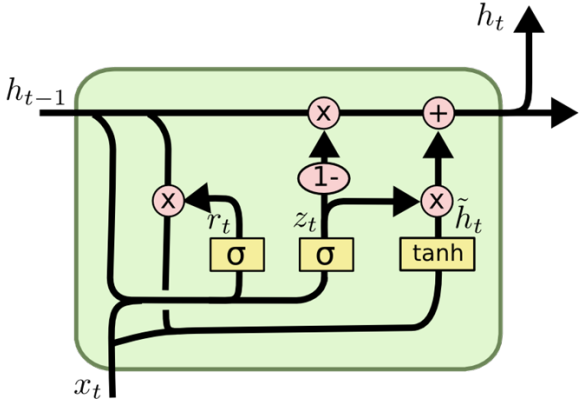




Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

GRU

- A simpler version of LSTM introduced by Cho et al. in 2014
- Fewer gates, less computation
- Replace the forget gate (f_t) with $(1 - z_t) * h_{t-1}$
- On each timestep t we have input and hidden state (no cell state).



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$: input gate

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$: reset gate

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$: new hidden state content

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$: hidden state

How LSTM solves vanishing gradient / long-term dependency?

The LSTM architecture makes it easier for the RNN to preserve information over many timesteps

- e.g., if the forget gate is set to 1 for a cell dimension and the input gate set to 0, then the information of that cell is preserved indefinitely.
- By contrast, it's harder for traditional RNNs to learn a recurrent weight matrix W_h that preserves info in hidden state.

Open questions: does LSTM solve the vanishing/exploding gradient problem? Maybe because of its gated design.

Refer to Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

Apply RNN-LM: text generation

- Just like a uni-gram / n-gram Language Model, you can use an RNN Language Model to generate text by repeated sampling. Sampled output is next step's input.

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done. ...

An example generated by Obama-RNN
Source: https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

"The Malfoys!" said Hermione.

Harry was watching him. He looked like Madame Maxime. When she strode up the wrong staircase to visit himself...

A Harry Potter chapter generated by LSTM-RNN
Source: https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

More examples with RNN-LM

- Recipe generator: https://gist.github.com/nylki/1efbaa36635956d35bcc
  - Follow the typical format of a recipe
  - RNN has the ability to learn the format of materials

```
      Title: CHOCOLATE RANCH BARBECUE
Categories: Game, Casseroles, Cookies, Cookies
      Yield: 6 Servings

      2 tb Parmesan cheese -- chopped
      1 c  Coconut milk
      3    Eggs, beaten
```

- Color name generator: https://colornamer.robertcooper.me/
  - a character-level RNN-LM (predicts what character comes next)

Ghasty Pink 231 137 165
Power Gray 151 124 112
Navel Tan 199 173 140
Bock Coe White 221 215 236

Sand Dan 201 172 143
Grade Bat 48 94 83
Light Of Blast 175 150 147
Grass Bat 176 99 108

Evaluating language models

- The standard evaluation metrics for LM is perplexity.

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

- Perplexity can be derived directly from the cross-entropy loss $J(\theta)$:

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} - \log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)} \right) = \exp(J(\theta))$$

- Lower perplexity is better.

Model performance comparison (1)

- Compare models with different history
- Train models
  - 38 million words – Wall Street Journal
- Compute perplexity on held-out test set
  - 1.5 million words (~20K unique, smoothed)
- N-gram Order | Perplexity
  - Unigram | 962
  - Bigram | 170
  - Trigram | 109

Perplexity improves (lower is better)

Model performance comparison (2)

n-gram model →

Increasingly complex RNNs ↓

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

↓ Perplexity improves (lower is better)

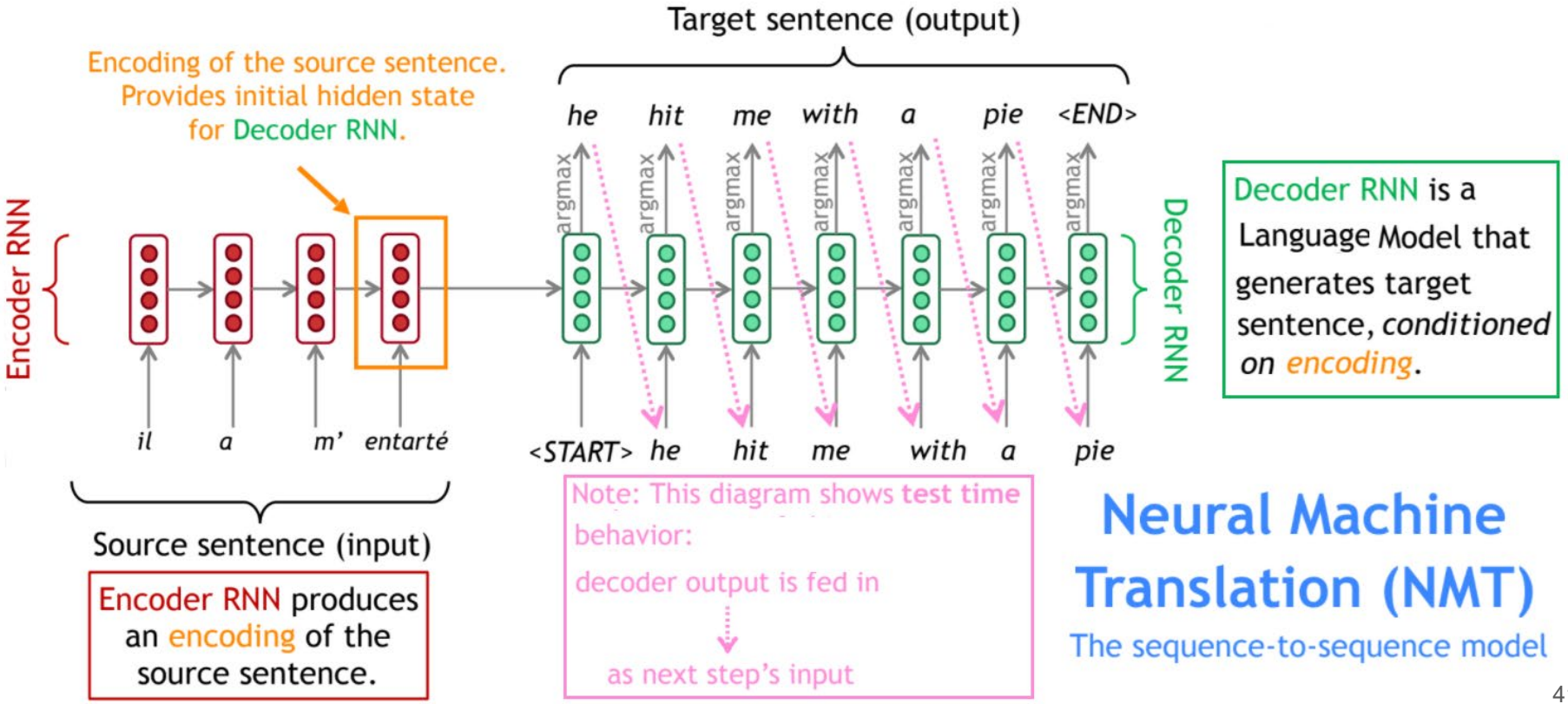
LSTM < RNN < n-gram

https://engineering.fb.com/2016/10/25/ml-applications/building-an-efficient-neural-language-model-over-a-billion-words/

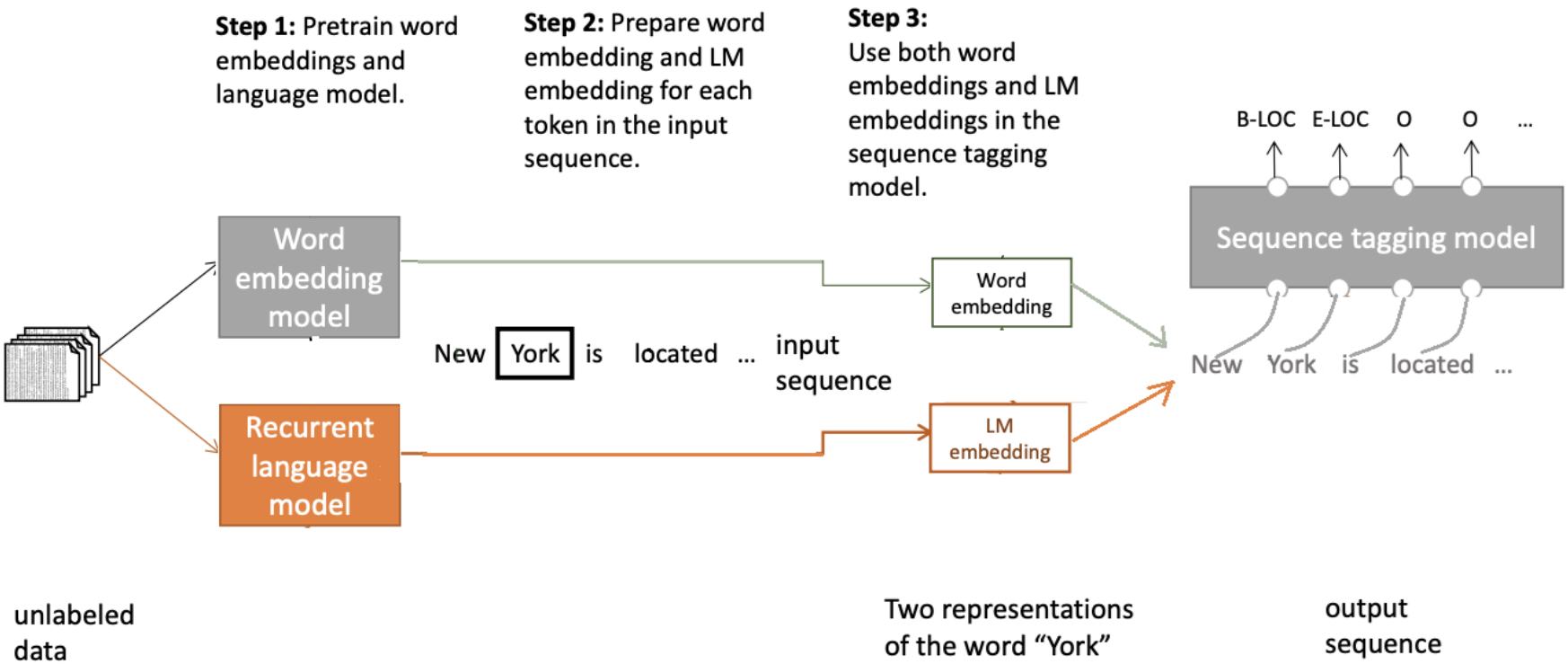
Model performance comparison (3)

https://paperswithcode.com/sota/language-modelling-on-wikitext-103

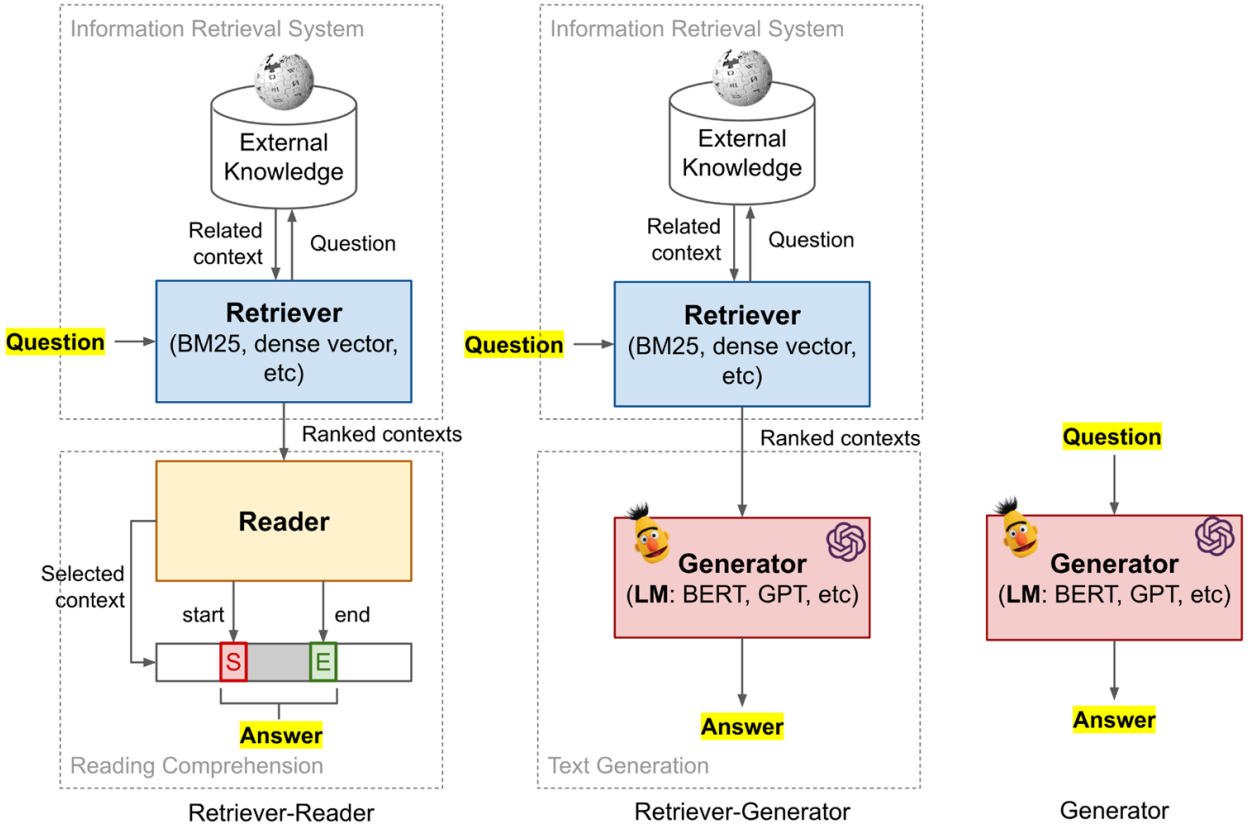
LM in more complex tasks – machine translation



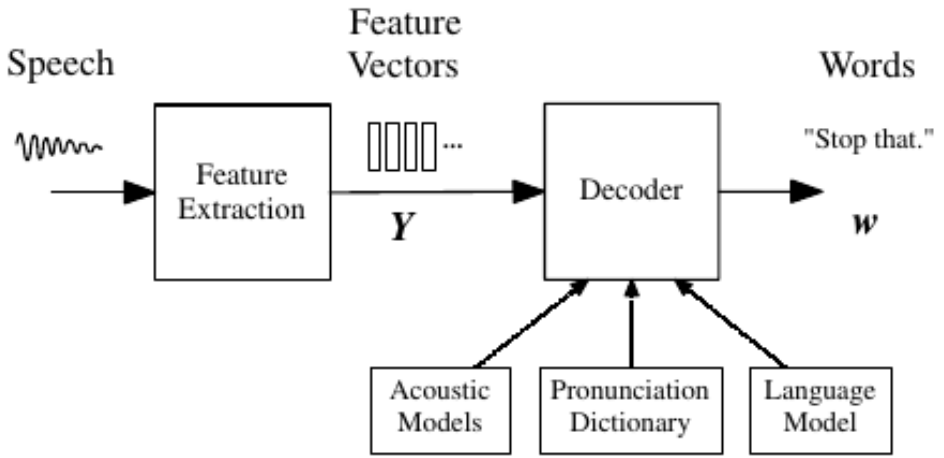
LM in more complex tasks - NER

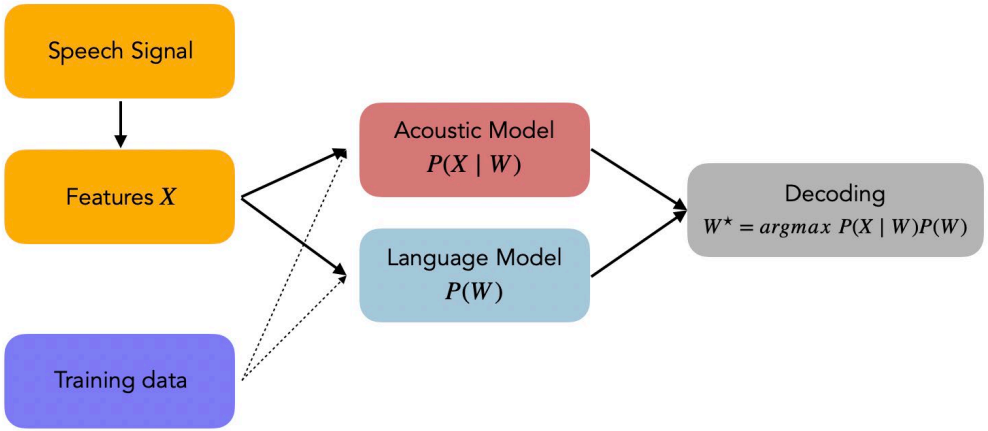


LM in more complex tasks - QA



LM in more complex tasks - speech recognition





Thanks for your attention!