344.175 VL: Natural Language Processing
n-Gram Language Models

Navid Rekab-saz

Email: navid.rekabsaz@jku.at
Office hours: https://navid-officehours.youcanbook.me

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computational Perception

Agenda

- *n*-gram language models
- Count-based *n*-gram LM
- Neural *n*-gram LM

Agenda

- ***n*-gram language models**
- Count-based *n*-gram LM
- Neural *n*-gram LM

Some slides are adopted from http://web.stanford.edu/class/cs224n/ & https://web.stanford.edu/~jurafsky/slp3/3.pdf

Language Modeling

- Language Modeling is the task of predicting a word (or a subword or character) given a context:

$$P(v|\text{context})$$

- A Language Model (LM) can answer the questions like

$$P(v|\textit{the students opened their})$$

Language Modeling – formal definition

- Given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, a language model calculates the probability distribution of next word $x^{(t+1)}$ over all words in vocabulary

$$P(x^{(t+1)} | x^{(1)}, \ldots, x^{(t-1)}, x^{(t)})$$

x is any word in the vocabulary $\mathbb{V} = \{v1, v2, \ldots, vN\}$

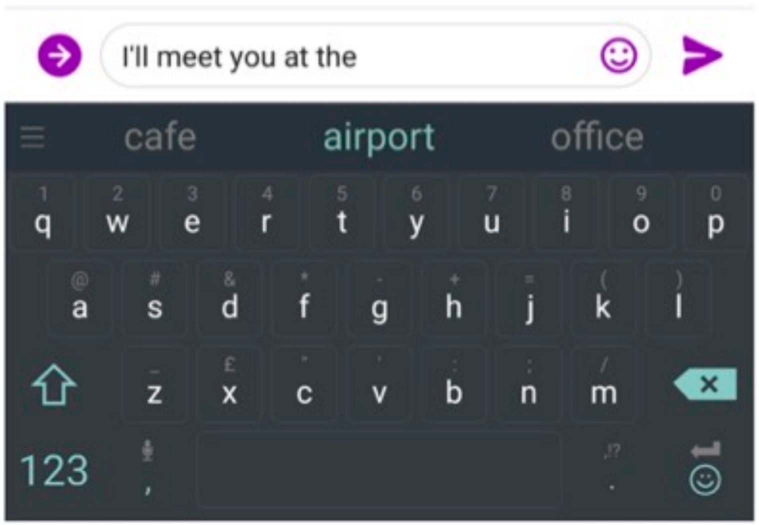
☞ Note: this is the definition of directed left-to-right language models.

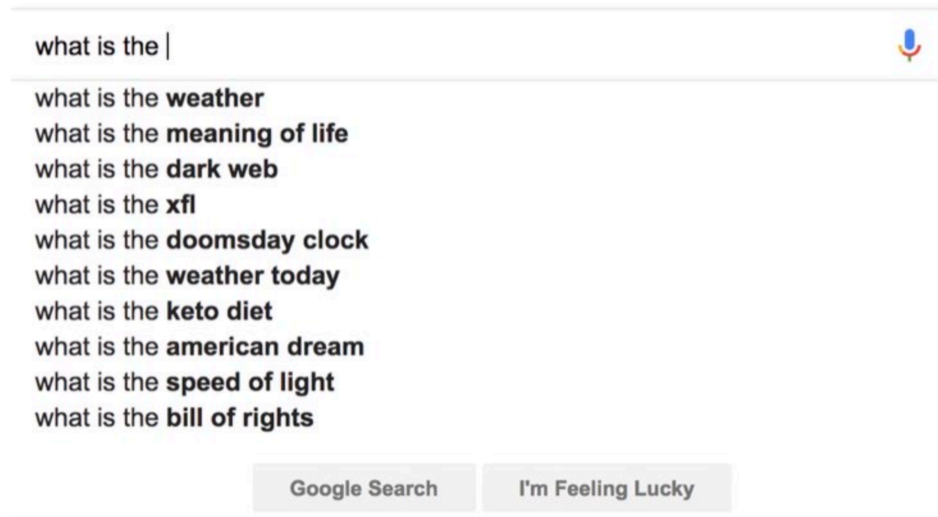
Why Language Modeling?

- Language Modeling is a benchmark task that helps us measure our progress on understanding language

- LMs are a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
  - Predictive typing
  - Spelling/grammar correction
  - Automatic speech recognition (ASR)
  - Handwriting recognition
  - Machine translation
  - Summarization
  - Dialogue /chatbots
  - etc.

Direct usages for next word prediction





[link]

Probability of a Text

- A Language Model can also assign probability to the validity a piece of text
  - How probable it is that a sentence appears in a language.

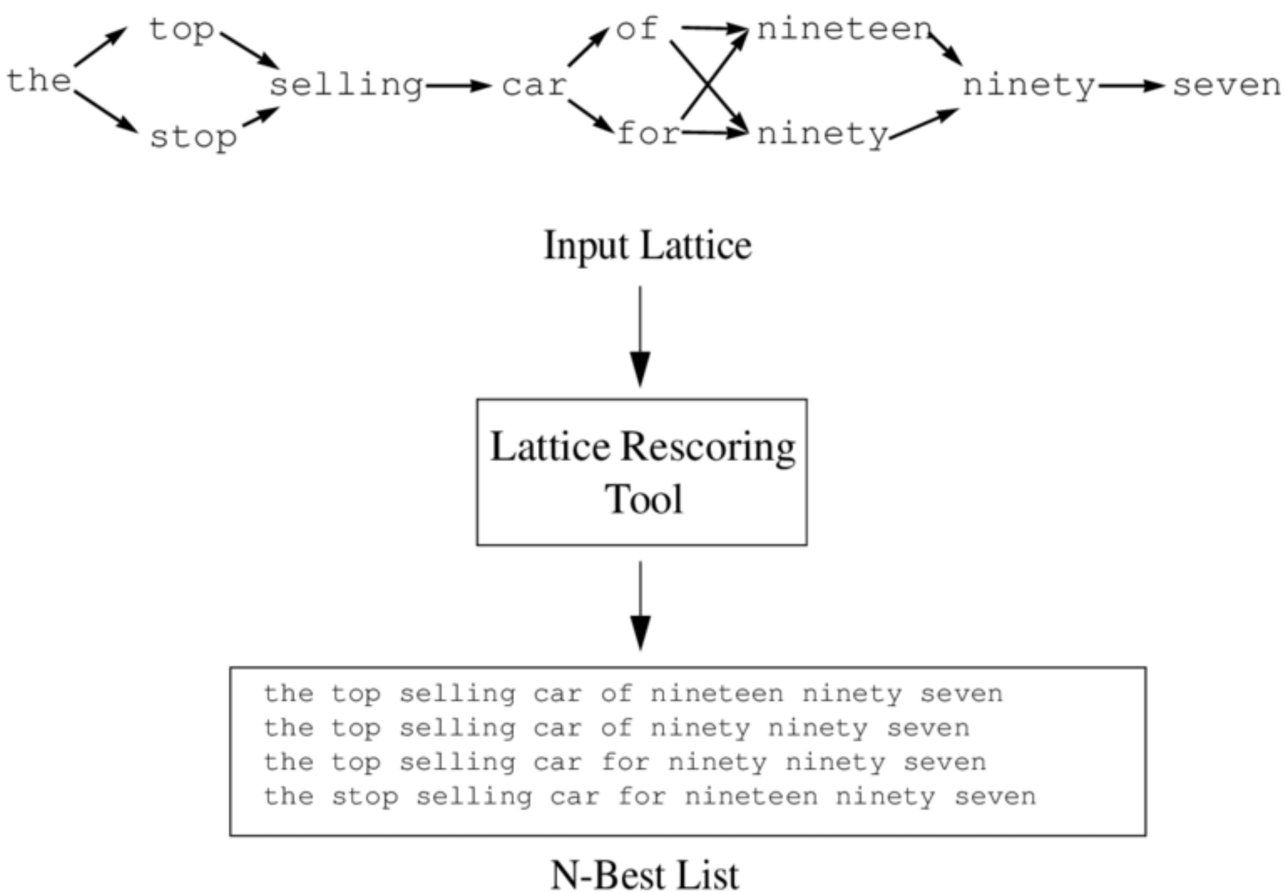
$$P\left(x^{(1)}, \ldots, x^{(T)}\right) = ?$$

- According to a (directed left-to-right) Language Model, the probability of a given text is computed by:

$$P\left(x^{(1)}, \ldots, x^{(T)}\right) = P\left(x^{(1)}\right) \times P\left(x^{(2)} \middle| x^{(1)}\right) \times \cdots \times P\left(x^{(T)} \middle| x^{(1)}, \ldots, x^{(T-1)}\right)$$

$$P\left(x^{(1)}, \ldots, x^{(T)}\right) = \prod_{t=1}^{T} P(x^{(t)} | x^{(1)}, \ldots, x^{(t-1)})$$

Usage in Automatic Speech Recognition (ASR)



n-gram Language Model

- Recall: a *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"*

- A *n*-gram Language Model collects frequency statistics of different *n*-grams in a corpus, and use these to calculate probabilities

N-gram LM as a conditional probability

- Markov assumption: decision at time t depends only on the current state

- In *n*-gram Language Model: predicting $x^{(t+1)}$ depends on preceding *n-1* words

- Without Markovian assumption:

$$P(x^{(t+1)}|x^{(1)}, \ldots, x^{(t-1)}, x^{(t)})$$

- *n*-gram Language Model:

$$P(x^{(t+1)}|\underbrace{x^{(t-n+2)}, \ldots, x^{(t-1)}, x^{(t)}}_{n\text{-}1 \text{ words}})$$

Agenda

- *n*-gram language models
- **Count-based *n*-gram LM**
- Neural *n*-gram LM

Some slides are adopted from http://web.stanford.edu/class/cs224n/ & https://web.stanford.edu/~jurafsky/slp3/3.pdf

n-gram LM using term counts

- Based on definition of conditional probability:

$$P\left(x^{(t+1)} \middle| x^{(t-n+2)}, \dots, x^{(t)}\right) = \frac{P\left(x^{(t-n+2)}, \dots, x^{(t)}, x^{(t+1)}\right)}{P\left(x^{(t-n+2)}, \dots, x^{(t)}\right)}$$

- The *n*-gram probability is calculated by counting *n*-grams and [*n–1*]-grams in a large corpus of text:

$$P\left(x^{(t+1)} \middle| x^{(t)}, \dots, x^{(t-n+2)}\right) \approx \frac{\mathrm{count}\left(x^{(t-n+2)}, \dots, x^{(t)}, x^{(t+1)}\right)}{\mathrm{count}\left(x^{(t-n+2)}, \dots, x^{(t)}\right)}$$

Example

- Example: learning a 4-gram Language Model

$\cancel{\textit{as the exam clerk started the clock, the}}\ \underbrace{\textit{students opened their}}_{\text{condition on this}}\ \_\_\_\_\_\_$

$$P(v|\textit{students opened their}) = \frac{P(\textit{students opened their } v)}{P(\textit{students opened their})}$$

- 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

Example – a bigram LM

- Trained on the data of a restaurant dialogue system

Bigram counts:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Bigram LM:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Count-based *n*-gram LMs – limitations

- **Sparsity**
  - What if the denominator never occurred in corpus?
    - Backoff: Probability is calculated for lower *n*-grams.
      - E.g., in „*students opened their v*“, if „*students opened their*“ does not exist, language model probability is calculated for “*opened their*”
      - Trigram is backed off to let a bigram do the job!
  - What if the nominator never occurred in corpus?
    - Approached by various smoothing methods

Laplace smoothing

- Add a small number like $\delta = 1$ to the count of all words:

$$P\left(x^{(t+1)} \middle| x^{(t-n+2)}, \dots, x^{(t)}\right) = \frac{\#\left(x^{(t-n+2)}, \dots, x^{(t)}, x^{(t+1)}\right)}{\#\left(x^{(t-n+2)}, \dots, x^{(t)}\ \right)}$$

$$P\left(x^{(t+1)} \middle| x^{(t-n+2)}, \dots, x^{(t)}\right) = \frac{\#\left(x^{(t-n+2)}, \dots, x^{(t)}, x^{(t+1)}\right)}{\sum_{v \subset V} \#\left(x^{(t-n+2)}, \dots, x^{(t)}, v\right)}$$

$$P_{Laplace}\left(x^{(t+1)} \middle| x^{(t-n+2)}, \dots, x^{(t)}\right) = \frac{\#\left(x^{(t-n+2)}, \dots, x^{(t)}, x^{(t+1)}\right) + 1}{\sum_{v \subset V}\left(\#\left(x^{(t-n+2)}, \dots, x^{(t)}, v\right) + 1\right)}$$

$$P_{Laplace}\left(x^{(t+1)} \middle| x^{(t-n+2)}, \dots, x^{(t)}\right) = \frac{\#\left(x^{(t-n+2)}, \dots, x^{(t)}, x^{(t+1)}\right) + 1}{\sum_{v \subset V}\left(\#\left(x^{(t-n+2)}, \dots, x^{(t)}, v\right)\right) + |V|}$$

$$P_{Laplace}\left(x^{(t+1)} \middle| x^{(t-n+2)}, \dots, x^{(t)}\right) = \frac{\#\left(x^{(t-n+2)}, \dots, x^{(t)}, x^{(t+1)}\right) + 1}{\#\left(x^{(t-n+2)}, \dots, x^{(t)}\right) + |V|}$$

Laplace smoothing example

Original bigram counts:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Bigram counts added by 1:

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Laplace smoothing example

LM without smoothing:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

LM with Laplace smoothing:

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Count-based *n*-gram LMs – limitations

- **Sparsity**
  - What if the denominator never occurred in corpus?
    - Backoff: Probability is calculated for lower *n*-grams.
      - E.g., in „*students opened their* v“, if „*students opened their*“ does not exist, language model probability is calculated for “*opened their*”
      - Trigram is backed off to let a bigram do the job!
  - What if the nominator never occurred in corpus?
    - Approached by various smoothing methods
  - Sparsity issue becomes even more prominent in higher *n*-gram!

Learn more: https://web.stanford.edu/~jurafsky/slp3/3.pdf

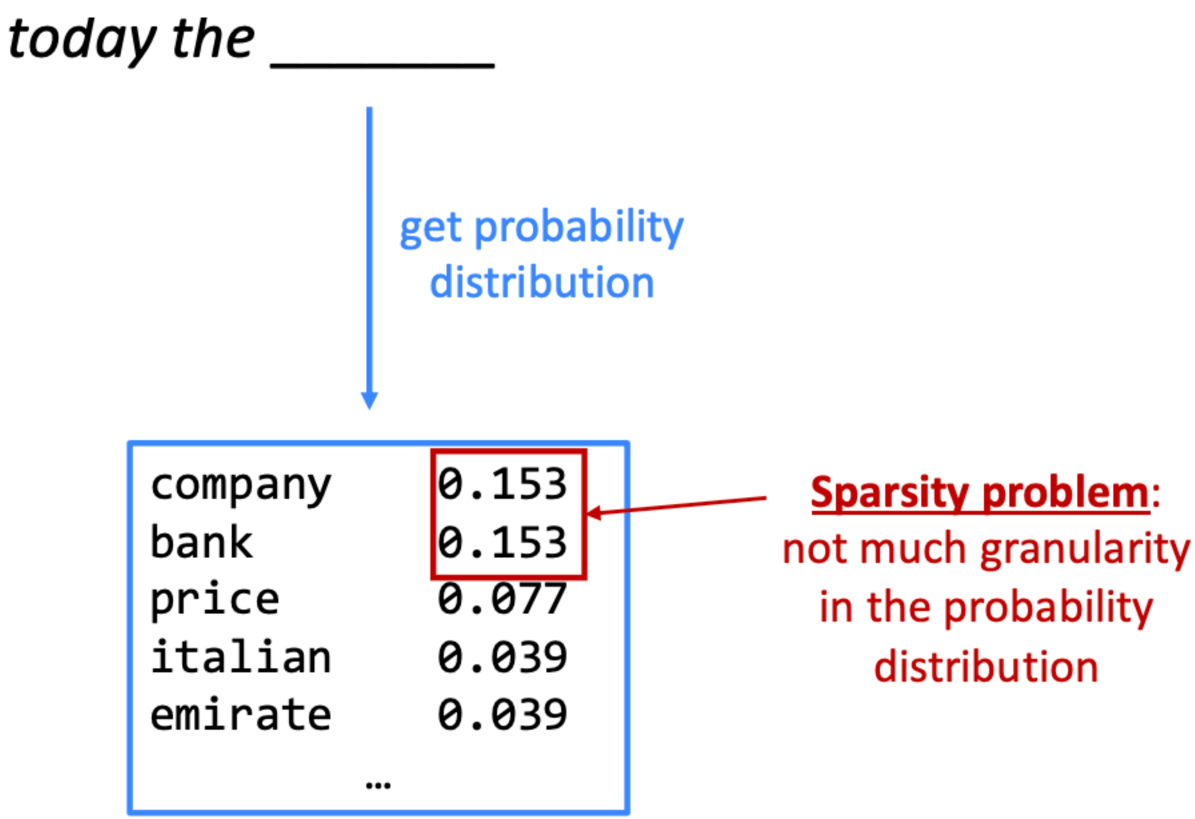
Count-based *n*-gram LMs – limitations

- **Storage**
  - An *n*-gram language model needs to store all levels of *n*-grams, from uni- to *n*-gram, observed in the corpus
  - Increasing *n* radically worsens the storage problem!

- **No understanding of tokens relations**
  - Semantic and syntactic relations between words are fully ignored
    - *"book"* and "*books*" or *"car"* and "*automobile*" are treated completely separately

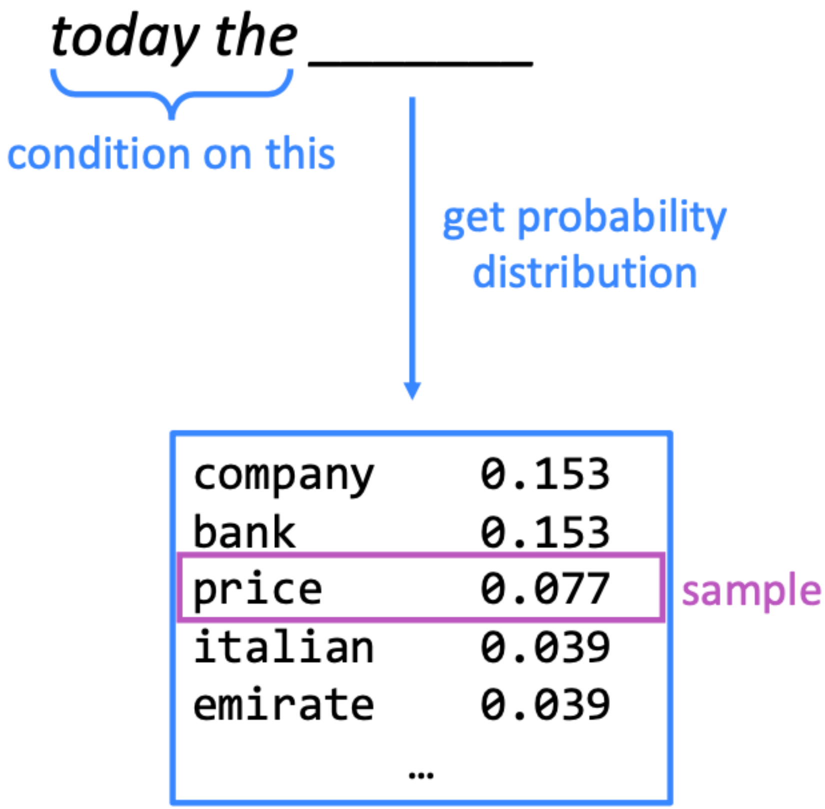
Generating text

- A trigram LM trained on Reuters corpus (1.7 M words)



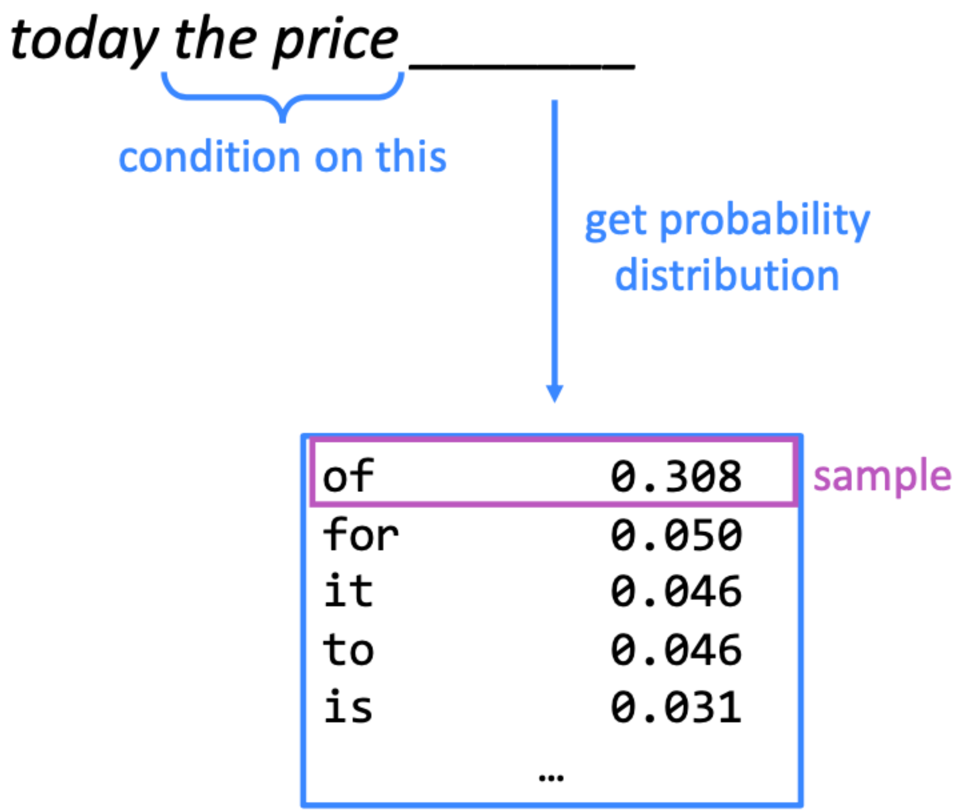
Generating text

- Generating text by sampling from the probability distributions



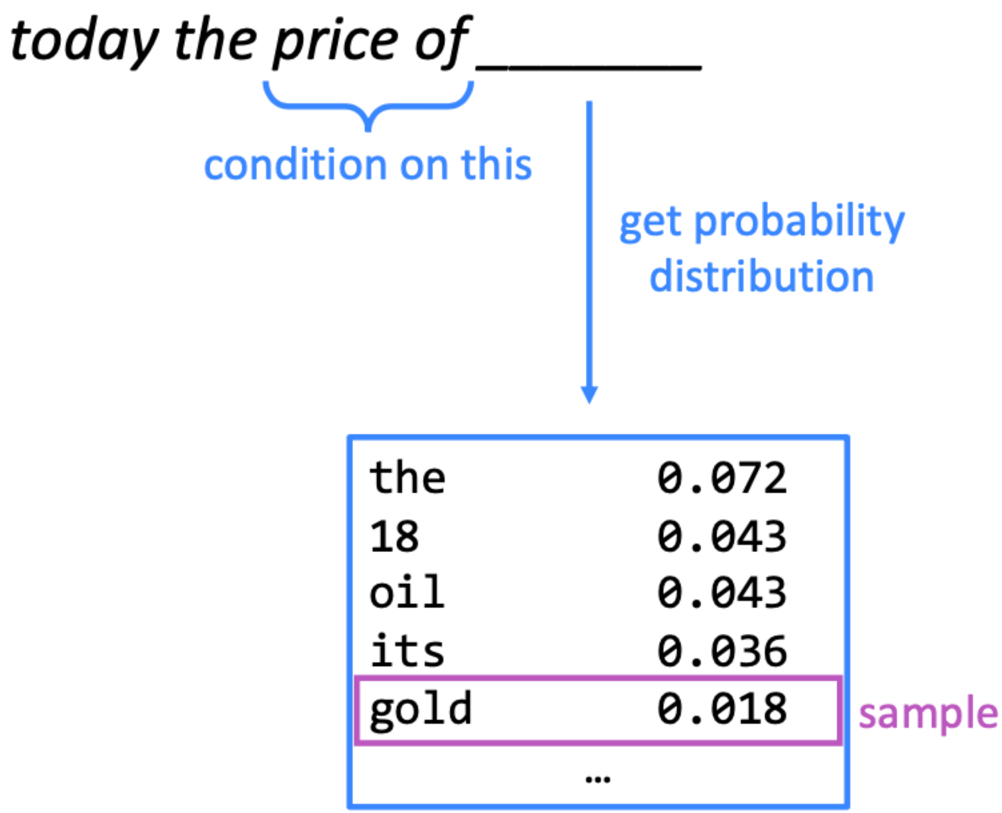
Generating text

- Generating text by sampling from the probability distributions



Generating text

- Generating text by sampling from the probability distributions



Generating text

- Generating text by sampling from the probability distributions

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 .

- Decently good in syntax … but incoherent!
- Increasing *n* makes the text more coherent but also intensifies the discussed issues

Generating text

- *n*-gram LMs trained on Shakespeare's works

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

Generating text

- *n*-gram LMs trained on Wall Street Journal

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Agenda

- *n*-gram language models
- Count-based *n*-gram LM
- **Neural *n*-gram LM**

n-gram language modeling with neural networks

Recall

- The aim of a *n*-gram Language Model is to calculate:

$$P\left(x^{(t+1)} \middle| x^{(t-n+2)}, \ldots, x^{(t)}\right)$$

- We can use a feed forward neural network to estimate this probability
- Immediate benefits:
  - Smooth probability estimation
  - Exploiting the semantic space of word embeddings (probably better generalization)

Basic idea from: Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, *3*(Feb), 1137-1155.

Neural *n*-gram LM – preparing training data

- Preparing training data for a neural 4-gram Language Model in the form of $(\text{context}, \text{next word})$, namely $(x^{(t-2)}x^{(t-1)}x^{(t)}, x^{(t+1)})$:

- For a given text corpus:

```
a fluffy cat sunbathes on the bank of river …
```

- Training data items would be:

```
(<bos> <bos> <bos>, a)
(<bos> <bos> a, fluffy)
(<bos> a fluffy, cat)
```

`(a fluffy cat, sunbathes)`

```
(fluffy cat sunbathes, on)
(cat sunbathes on, the)
(sunbathes on the, bank)
```

…

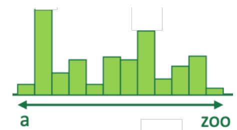
<bos> is a special token added to dictionary, referring to *beginning of sentence*

Also, dot is commonly replaced with <eos> token – *end of sentence*

Neural *n*-gram Language Model – architecture

$$P\left(x^{(t+1)} \middle| \text{a fluffy cat}\right) = \hat{y}^{(t)}_{x^{(t+1)}}$$

$$P(\text{sunbathes}|\text{a fluffy cat}) = \hat{y}^{(t)}_{\text{sunbathes}}$$



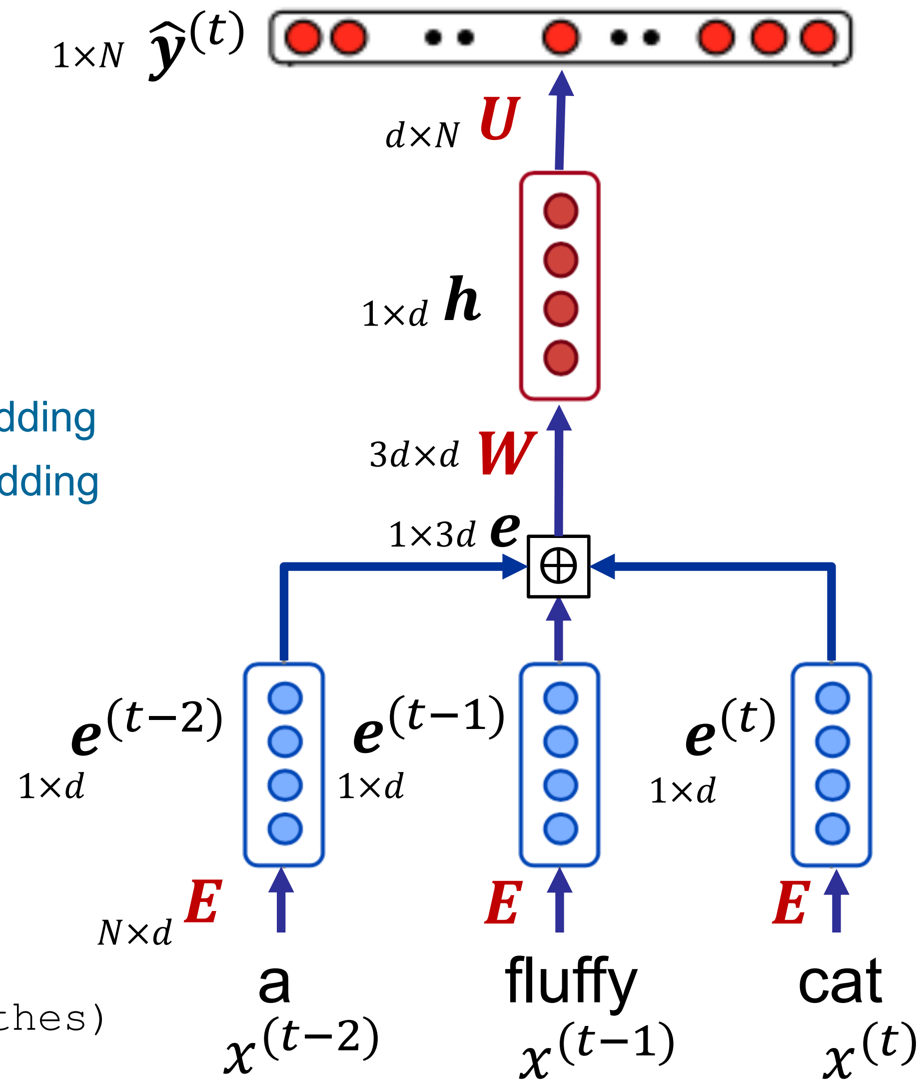
- $\boldsymbol{E} \rightarrow N{\times}d$
- $\boldsymbol{W} \rightarrow d{\times}((n-1){\times}d)$
- $\boldsymbol{U} \rightarrow d{\times}N$

$\boldsymbol{E}$ is called encoder embedding

$\boldsymbol{U}$ is called decoder embedding or output projection

N size of vocabulary

d embeddings dimension



A data item in training data:
`(a fluffy cat, sunbathes)`

N size of vocabulary
d embeddings dimension
$(n-1)$ number of preceding words
Parameters are shown in red

Formulation

Encoder

- From word to word embedding:
  - One-hot vector of word $x^{(t)}$ is $\boldsymbol{x}^{(t)}$ vector: $\boldsymbol{x}^{(t)} \rightarrow 1 \times N$
    - In $\boldsymbol{x}^{(t)}$, all values are 0 and only the value corresponding to the word $x^{(t)}$ is set to 1
  - Fetching word embedding: $\boldsymbol{e}^{(t)} = \boldsymbol{x}^{(t)} \boldsymbol{E}$ $\quad \boldsymbol{e}^{(t)} \rightarrow 1 \times d$
    - In practice, $\boldsymbol{e}^{(t)}$ is achieved by fetching the vector of $x^{(t)}$ from $\boldsymbol{E}$. No need for $\boldsymbol{x}^{(t)}$ in practice
- Concatenation of $(n-1)$ word embeddings:

$$\boldsymbol{e} = [\boldsymbol{e}^{(t-2)}, \boldsymbol{e}^{(t-1)}, \boldsymbol{e}^{(t)}] \qquad \boldsymbol{e} \rightarrow 1 \times (n-1)d$$

- Hidden layer: $\boldsymbol{h} = \tanh(\boldsymbol{e}\boldsymbol{W} + \boldsymbol{b}^{w}) \qquad \boldsymbol{h} \rightarrow 1 \times d$

$\boldsymbol{E} \rightarrow N \times d$
$\boldsymbol{W} \rightarrow (n-1)d \times d$
$\boldsymbol{b}^{w} \rightarrow 1 \times d$

N size of vocabulary
d embeddings dimension
$(n-1)$ number of preceding words
Parameters are shown in red

Formulation

Decoder

- Predicted logits:

$$\boldsymbol{z} = \boldsymbol{h}\boldsymbol{U} + \boldsymbol{b}^u \qquad \boldsymbol{z} \rightarrow 1 \times N$$

- Predicted probability distribution:

$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{z}) \qquad \hat{\boldsymbol{y}}^{(t)} \rightarrow 1 \times N$$

- Probability of any next word v at step t:

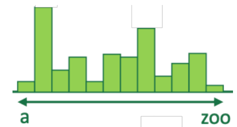
$$P\left(v \middle| x^{(t)}, \dots, x^{(t-n+2)}\right) = \hat{y}_v^{(t)}$$

$\boldsymbol{U} \rightarrow d \times N$
$\boldsymbol{b}^u \rightarrow 1 \times N$

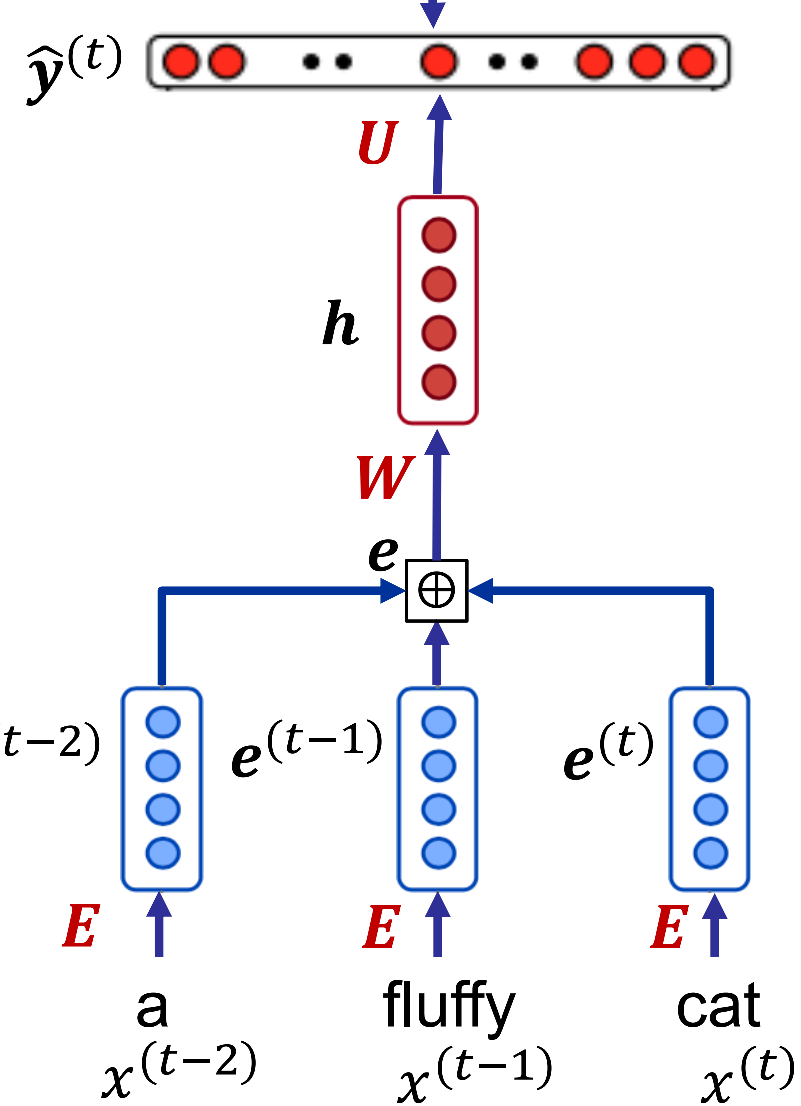
Loss function

$$P\left(x^{(t+1)} \middle| \text{a fluffy cat}\right) = \hat{y}^{(t)}_{x^{(t+1)}}$$

$$P(\text{sunbathes}|\text{a fluffy cat}) = \hat{y}^{(t)}_{\text{sunbathes}}$$



$$\mathcal{L} = -\log \hat{y}^{(t)}_{x^{(t+1)}} = -\log \hat{y}^{(t)}_{\text{sunbathes}}$$



A data item in training data:
`(a fluffy cat, sunbathes)`

Training *n*-gram neural LM

- Start with a large text corpus: $x^{(1)}, \dots, x^{(T)}$
- In every step t, give *n-1* previous words as input and output the predicted probability distribution of the next words $\hat{\boldsymbol{y}}^{(t)}$
- Loss function at t is Negative Log Likelihood of the predicted probability for the actual next word $x^{(t+1)}$

$$\mathcal{L}^{(t)} = -\log P\left(x^{(t+1)} \middle| x^{(t-n+2)}, \dots, x^{(t)}\right)$$

$$\mathcal{L}^{(t)} = -\log \hat{y}^{(t)}_{x^{(t+1)}}$$

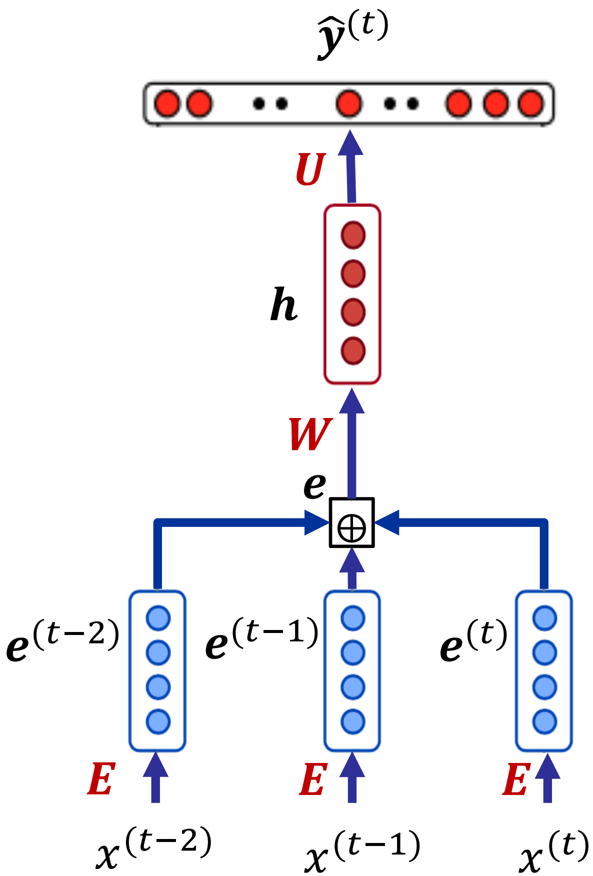
- Overall loss is the average over all time steps:
  - In practice, loss is calculated over batches of text chunks

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} \mathcal{L}^{(t)}$$

About transfer learning

- $\boldsymbol{E}$ provides a vector for each word in dictionary
- We can initialize (some) vectors of $\boldsymbol{E}$ with pre-trained word embeddings like GloVe or word2vec
  - In this case, for every word in $\boldsymbol{E}$ we fetch the corresponding vector from a pre-trained word embedding model
  - If the word doesn't exist, as before, its vector is randomly initialized
- We can also do the same for $\boldsymbol{U}$
- This *better* initialization of parameters is a form of transfer learning

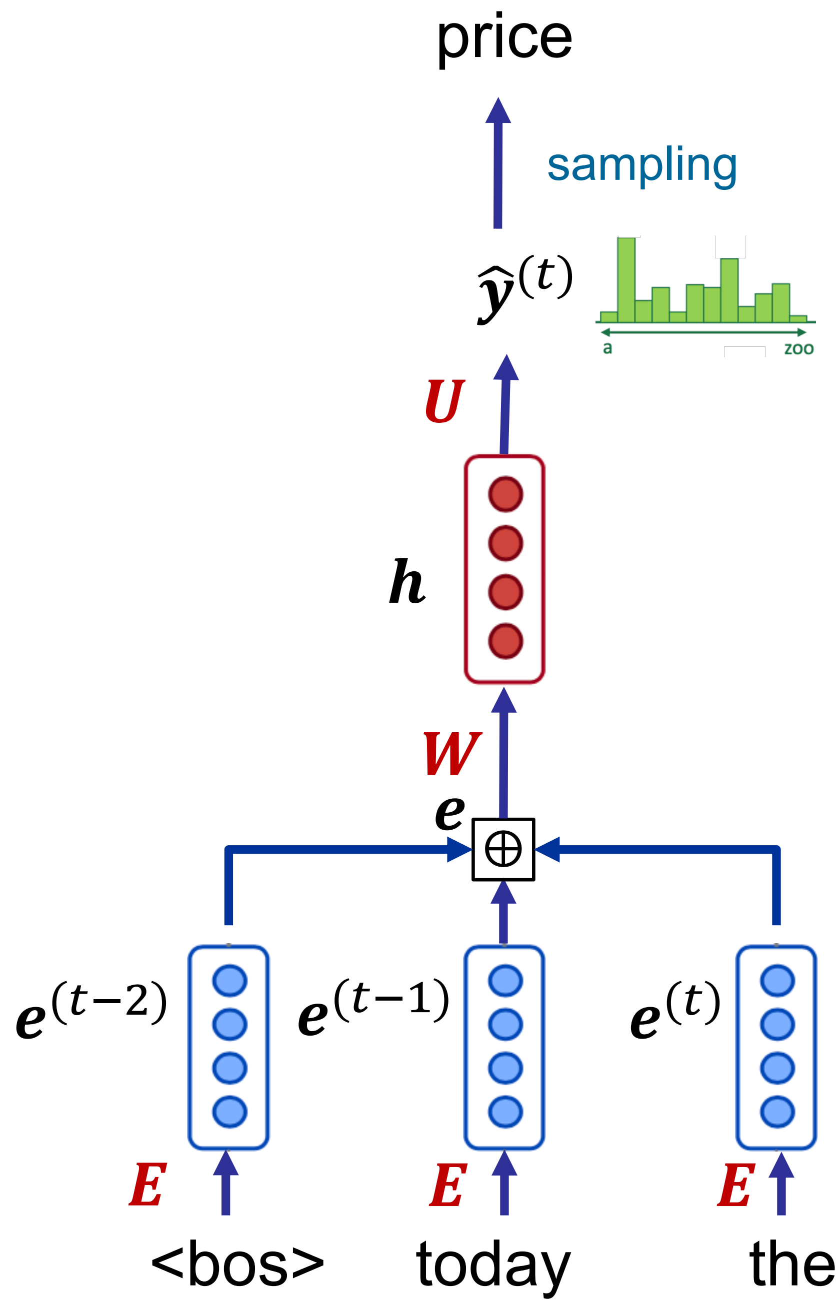
☞ Even without transfer learning, after training the LM, the $\boldsymbol{E}$ and $\boldsymbol{U}$ matrices provide *proper* word embeddings which can be used independently



Generating text

text seed:
today the

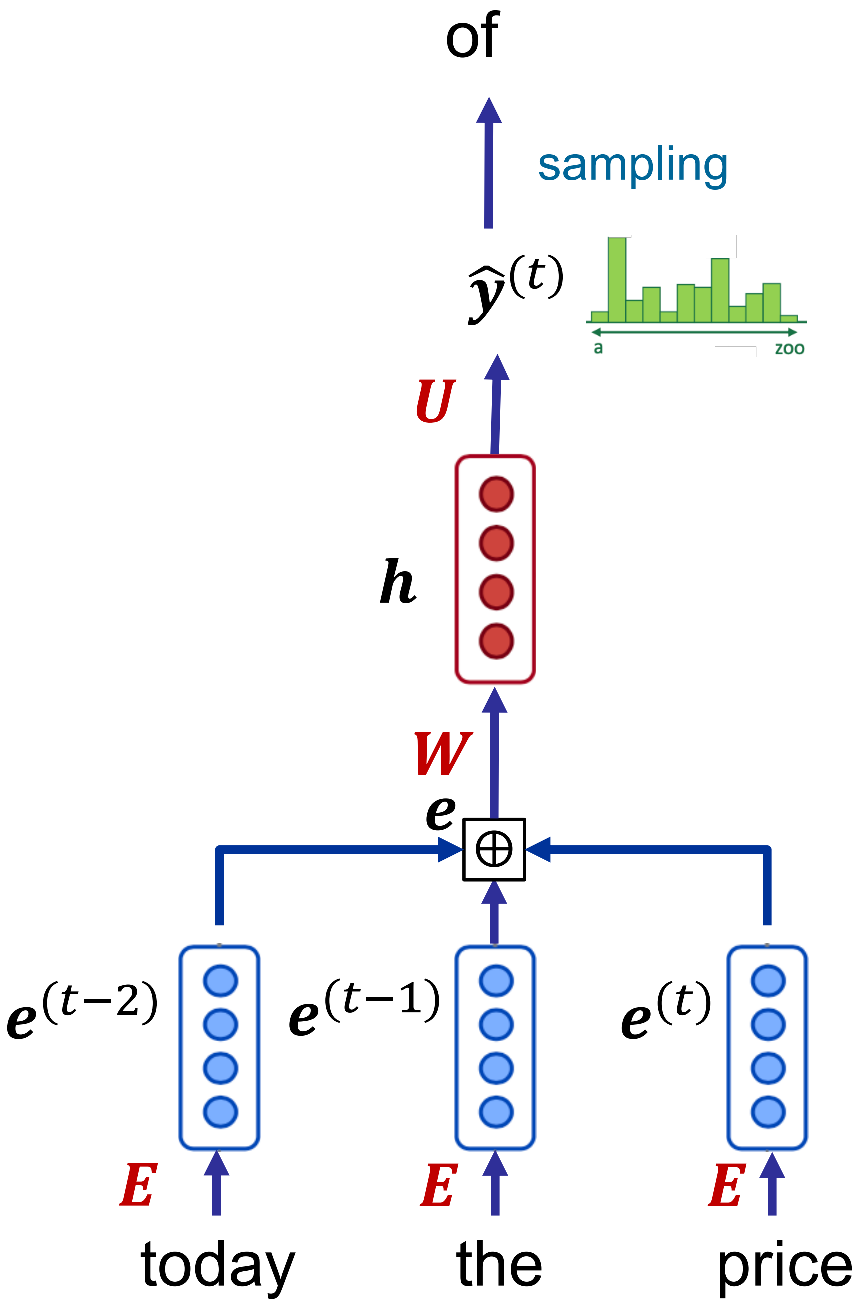
after generation:
today the price



Generating text

(before):
today the price

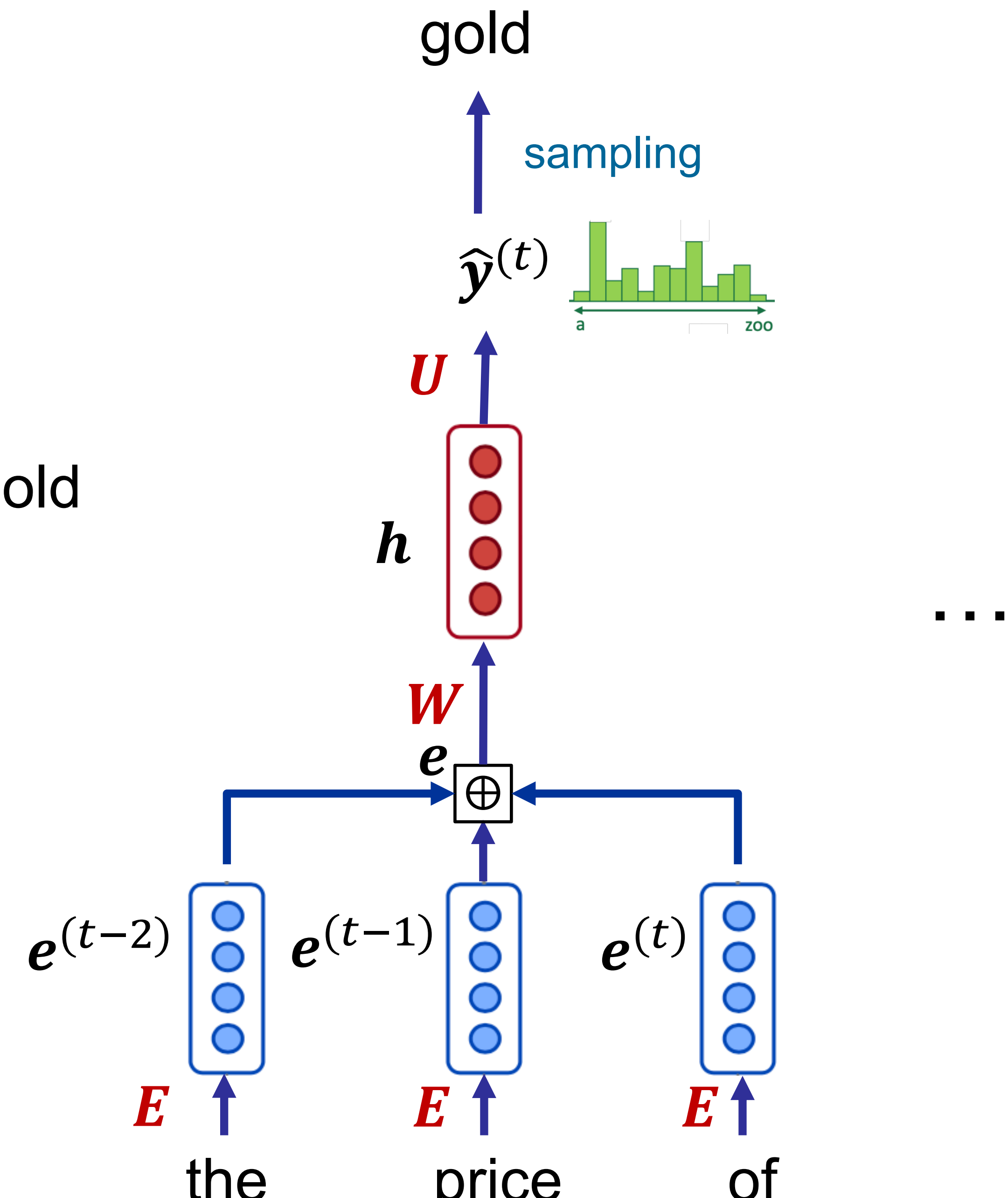
(after):
today the price of



Generating text

(before):
today the price of

(after):
today the price of gold



Neural n-gram LMs – summary

- Neural n-gram LMs <u>predict</u> next word probabilities

- Neural n-gram LMs benefit from semantic relations of words, provided by the encoder and decoder embeddings

- Neural n-gram LMs provide a smooth probability distribution

- At inference time, neural n-gram LMs require a forward pass
  - Count-based n-gram LMs might be more convenient at inference time in practice, since they calculate probabilities simply from the stored counts