Natural Language Processing with Deep Learning
CS224N/Ling284

Christopher Manning

Lecture 1: Introduction and Word Vectors

ChatGPT, GPT-4, and more

OpenAI

Hey please draft a polite mail to explain my boss Jeremy that I would not be able to come to office for next 2 days because my 9 year song Peter is angry with me that I am not giving him much time. So I have plans to take him out for a trip. I will return and complete my work by meeting deadline.

Dear Jeremy,

I hope this email finds you well. I'm writing to let you know that I will not be able to come into the office for the next two days.

My 9-year-old son Peter is feeling upset with me because he feels like I haven't been giving him enough attention lately. In order to make it up to him, I've decided to take him on a little trip to help us reconnect.

I understand that this may cause some inconvenience, but I promise to make up for any lost time by working extra hard to meet any deadlines when I return.

Thank you for your understanding.

Best regards,
[Your Name]

User What is unusual about this image?

Source: Barnorama

GPT-4 The unusual thing about this image is that a man is ironing clothes on an ironing board attached to the roof of a moving taxi.

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:
hotel, conference, motel – a localist representation

Means one 1, the rest 0s

Such symbols for words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000+)

Problem with words as discrete symbols

Example: in web search, if a user searches for "Seattle motel", we would like to match documents containing "Seattle hotel"

But:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

These two vectors are orthogonal

There is no natural notion of **similarity** for one-hot vectors!

Solution:

- Could try to rely on WordNet's list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**

Representing words by their context

- Distributional semantics: **A word's meaning is given by the words that frequently appear close-by**
  - *"You shall know a word by the company it keeps"* (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- We use the many contexts of *w* to build up a representation of *w*

…government debt problems turning into ***banking*** *crises as happened in 2009…*
…saying that Europe needs unified ***banking*** *regulation to replace the hodgepodge…*
…India has just given its ***banking*** *system a shot in the arm…*

These context words will represent ***banking***

Word vectors

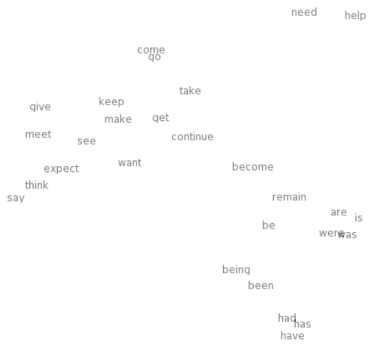
We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector dot (scalar) product

banking = $\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$ *monetary* = $\begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix}$

Note: word vectors are also called (word) embeddings or (neural) word representations
They are a distributed representation

Word meaning as a neural word vector – visualization

expect = $\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$

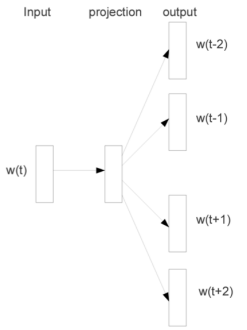


3. Word2vec: Overview

Word2vec is a framework for learning word vectors
(Mikolov et al. 2013)

Idea:

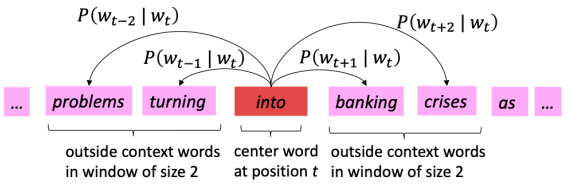
- We have a large corpus ("body") of text: a long list of words
- Every word in a fixed vocabulary is represented by a vector
- Go through each position *t* in the text, which has a center word *c* and context ("outside") words *o*
- Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa)
- Keep adjusting the word vectors to maximize this probability



Skip-gram model
(Mikolov et al. 2013)

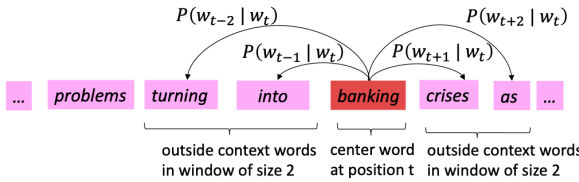
Word2Vec Overview

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2Vec Overview

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2vec: objective function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_t. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

- **Question:** How to calculate $P(w_{t+j} \mid w_t; \theta)$?
- **Answer:** We will *use two* vectors per word w:
  - v_w when w is a center word
  - u_w when w is a context word

} These word vectors are subparts of the big vector of all parameters θ

- Then for a center word c and a context word o:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Word2vec: prediction function

② Exponentiation makes anything positive

① Dot product compares similarity of *o* and *c*.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

③ Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \to (0,1)^n$ ← Open region

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in Deep Learning

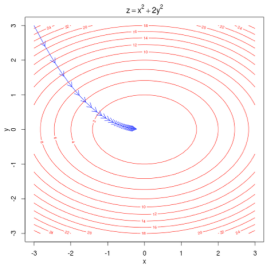
But sort of a weird name because it returns a distribution!

To train the model: Optimize value of parameters to minimize loss

To train a model, we gradually adjust parameters to minimize a loss

- Recall: θ represents **all** the model parameters, in one long vector
- In our case, with *d*-dimensional vectors and *V*-many words, we have →
- Remember: every word has two vectors

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



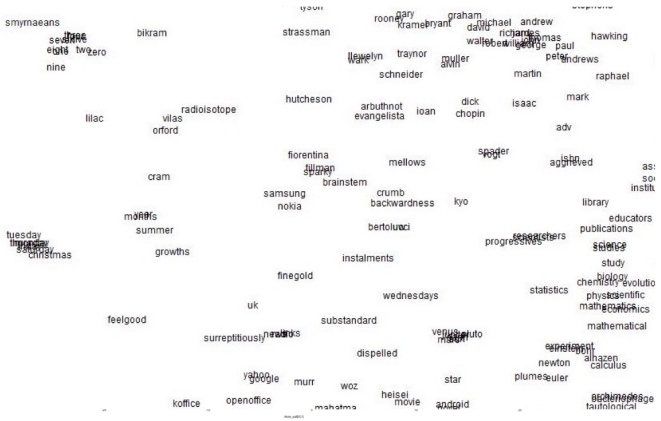
- We optimize these parameters by walking down the gradient (see right figure)
- We compute all vector gradients!

Natural Language Processing with Deep Learning
CS224N/Ling284

Christopher Manning

Lecture 2: Word Vectors, Word Senses, and Neural Classifiers

Word2vec maximizes objective by putting similar words nearby in space



4. Word2vec algorithm family: More details

[Mikolov et al. 2013: "Distributed Representations of Words and Phrases and their Compositionality"]

Why two vectors? → Easier optimization. Average both at the end

- But can implement the algorithm with just one vector per word … and it helps a bit

Two model variants:

1. Skip-grams (SG)
   Predict context ("outside") words (position independent) given center word
2. Continuous Bag of Words (CBOW)
   Predict center word from (bag of) context words

We presented: **Skip-gram model**

Loss functions for training:

1. Naïve softmax (simple but expensive loss function, when many output classes)
2. More optimized variants like hierarchical softmax
3. Negative sampling

So far, we explained **naïve softmax**

The skip-gram model with negative sampling

- The normalization term is computationally expensive (when many output classes):

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

A big sum over many words

- Hence, in standard word2vec, you implement the skip-gram model with **negative sampling**
- Idea: train binary logistic regressions to differentiate a true pair (center word and a word in its context window) versus several "noise" pairs (the center word paired with a random word)

8. Deep Learning Classification: Named Entity Recognition (NER)

- The task: find and classify names in text, by labeling word tokens, for example:

  Last night , Paris Hilton wowed in a sequin gown .
  PER PER
  Samuel Quinn was arrested in the Hilton Hotel in Paris in April 1989 .
  PER PER LOC LOC LOC DATE DATE

- Possible uses:
  - Tracking mentions of particular entities in documents
  - For question answering, answers are usually named entities
  - Relating sentiment analysis to the entity under discussion
- Often followed by Entity Linking/Canonicalization into a Knowledge Base such as Wikidata

Simple NER: Window classification using binary logistic classifier

- **Idea:** classify each word in its context window of neighboring words
- Train logistic classifier on hand-labeled data to classify center word {yes/no} for each class based on a concatenation of word vectors in a window
  - Really, we usually use multi-class softmax, but we're trying to keep it simple ☺
- **Example:** Classify "Paris" as +/– location in context of sentence with window length 2:

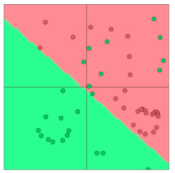
the museums in Paris are amazing to see .

$X_{window} = [\ x_{museums}\ \ x_{in}\ \ x_{Paris}\ \ x_{are}\ \ x_{amazing}\ ]^T$

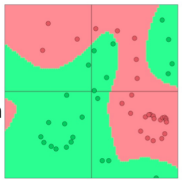
- Resulting vector $x_{window} = x \in R^{5d}$
- To classify all words: run classifier for each class on the vector centered on each word in the sentence

Neural classification

- Typical ML/stats softmax classifier: $p(y|x) = \frac{\exp(W_y.x)}{\sum_{c=1}^{C}\exp(W_c.x)}$
- Learned parameters θ are just elements of *W* (not input representation *x*, which has sparse symbolic features)
- Classifier gives linear decision boundary, which can be limiting



- A **neural network classifier** differs in that:
  - We learn **both *W* and (distributed!)** representations for words
  - The word vectors *x* re-represent one-hot vectors, moving them around in an intermediate layer vector space, for easy classification with a (linear) softmax classifier
    - Conceptually, we have an embedding layer: *x = Le*
  - We use deep networks—more layers—that let us re-represent and compose our data multiple times, giving a non-linear classifier



But typically, it is linear relative to the pre-final layer representation

NER: Binary neural classifier for center word being location

- We do supervised training and want high score if it's a location

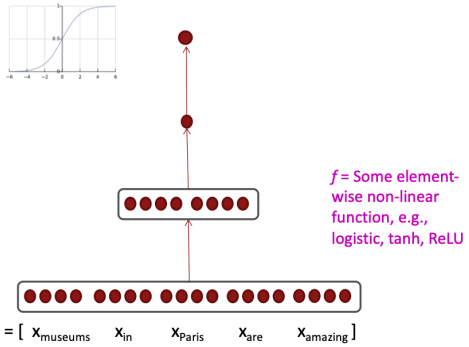
$$J_t(\theta) = \sigma(s) = \frac{1}{1+e^{-s}}$$

predicted model probability of class

$$s = \boldsymbol{u}^T\boldsymbol{h}$$

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

$$\boldsymbol{x} \quad \text{(input)}$$



f = Some element-wise non-linear function, e.g., logistic, tanh, ReLU

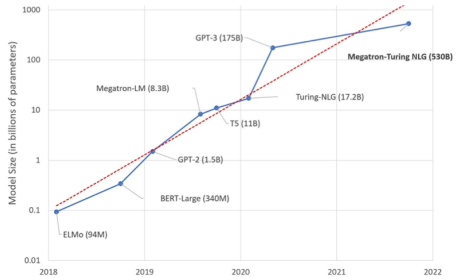
Natural Language Processing with Deep Learning CS224N/Ling284

Tatsunori Hashimoto

Lecture 5: Language Models and Recurrent Neural Networks

(Slides mostly from Chris Manning's 2023 version)

Modern neural networks (esp. language models) are *enormous*



- Large, deep neural nets are a cornerstone of modern NLP systems

https://huggingface.co/blog/large-language-models

But building large neural networks isn't easy or obvious

Greedy Layer-Wise Training of Deep Networks

Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle
Université de Montréal
Montréal, Québec
{bengioy,lamblinp,popovicd,larocheh}@iro.umontreal.ca

much less expressive than deep ones.

However, until recently, it was believed too difficult to train deep multi-layer neural networks. Empirically, deep networks were generally found to be not better, and often worse, than neural networks with one or two hidden layers (Tesauro, 1992). As this is a negative result, it has not been much reported in the machine learning literature. A reasonable explanation is that gradient-based optimization starting from random initialization may get stuck near poor solutions. An approach that has been explored with some success in the past is based on *constructively* adding layers. This was previously done using a

[Bengio et al 2006]

- It took a long time and much work to make deep neural networks practical!

2. Language Modeling

- **Language Modeling** is the task of predicting what word comes next

the students opened their _____ → *books*, *laptops*, *exams*, *minds*

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

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

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

- A system that does this is called a **Language Model**

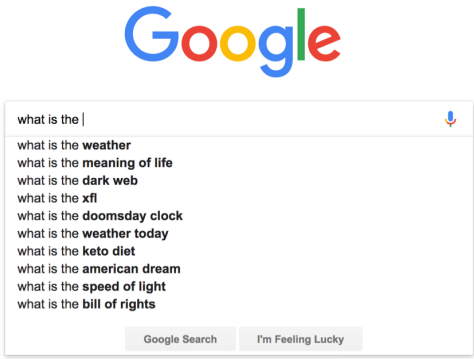
Language Modeling

- You can also think of a Language Model as a system that assigns a probability to a piece of text
- For example, if we have some text $x^{(1)}, \ldots, x^{(T)}$, then the probability of this text (according to the Language Model) is:

$$P(x^{(1)}, \ldots, x^{(T)}) = P(x^{(1)}) \times P(x^{(2)} | \, x^{(1)}) \times \cdots \times P(x^{(T)} | \, x^{(T-1)}, \ldots, x^{(1)})$$
$$= \prod_{t=1}^{T} P(x^{(t)} | \, x^{(t-1)}, \ldots, x^{(1)})$$

This is what our LM provides

You use Language Models every day!



n-gram Language Models

the students opened their _____

- **Question**: How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn an *n*-gram Language Model!
- **Definition:** An *n*-gram is a chunk of *n* consecutive words.
  - unigrams: "the", "students", "opened", "their"
  - bigrams: "the students", "students opened", "opened their"
  - trigrams: "the students opened", "students opened their"
  - four-grams: "the students opened their"
- **Idea:** Collect statistics about how frequent different n-grams are and use these to predict next word.

n-gram Language Models

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

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

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{P(x^{(t+1)}, x^{(t)}, \ldots, x^{(t-n+2)})}{P(x^{(t)}, \ldots, x^{(t-n+2)})}$$

(definition of conditional prob)

- **Question:** How do we get these *n*-gram and (*n*-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

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

(statistical approximation)

n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the~~ *students opened their* _____

discard

condition on this

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

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 Problems with n-gram Language Models

Sparsity Problem 1

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

(Partial) Solution: Add small δ to the count for every $w \in V$. This is called *smoothing*.

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

Sparsity Problem 2

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

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Note: Increasing *n* makes sparsity problems *worse*. Typically, we can't have *n* bigger than 5.

Storage Problems with n-gram Language Models

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

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

Increasing *n* or increasing corpus increases model size!

n-gram Language Models in practice

- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop*

Business and financial news

today the _____

get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

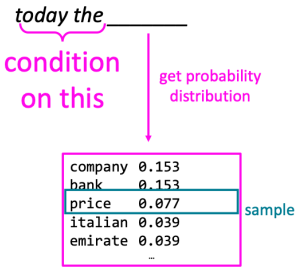
Sparsity problem: not much granularity in the probability distribution

Otherwise, seems reasonable!

*** Try for yourself:** https://nlpforhackers.io/language-models/

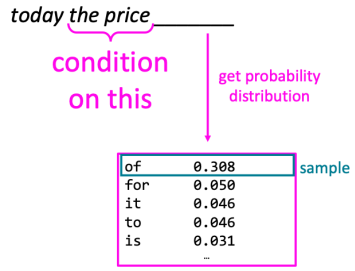
Generating text with a n-gram Language Model

You can also use a Language Model to generate text



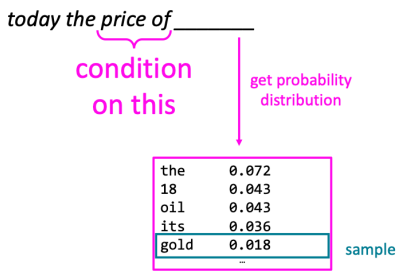
Generating text with a n-gram Language Model

You can also use a Language Model to generate text



Generating text with a n-gram Language Model

You can also use a Language Model to generate text



Generating text with a n-gram Language Model

You can also use a Language Model to generate text

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 .

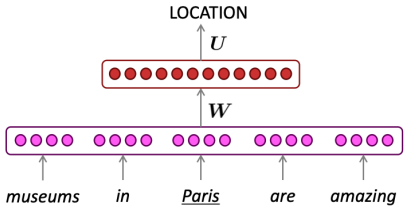
Surprisingly grammatical!

…but **incoherent.** We need to consider more than three words at a time if we want to model language well.

But increasing *n* worsens sparsity problem, and increases model size…

How to build a *neural* language model?

- Recall the Language Modeling task:
  - Input: sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$
  - Output: prob. dist. of the next word $P(x^{(t+1)} | x^{(t)}, \ldots, x^{(1)})$
- How about a window-based neural model?
  - We saw this applied to Named Entity Recognition in Lecture 2:



A fixed-window neural Language Model

~~as the proctor started the clock~~ the students opened their _____

discard

fixed window

A fixed-window neural Language Model

output distribution

$\hat{y} = \text{softmax}(Uh + b_2) \in \mathbb{R}^{|V|}$

hidden layer

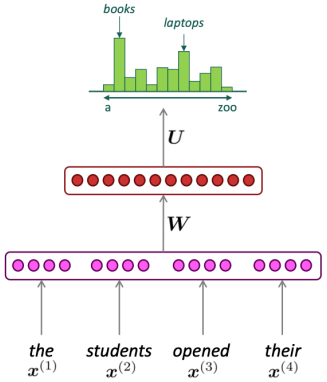
$h = f(We + b_1)$

concatenated word embeddings

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

words / one-hot vectors

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



A fixed-window neural Language Model

Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model

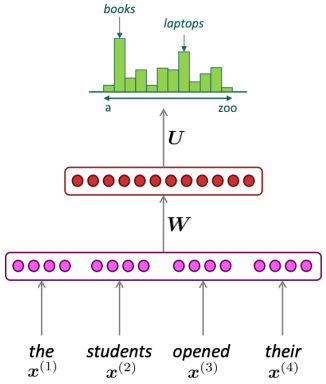
Improvements over *n*-gram LM:

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

Remaining **problems**:

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

We need a neural architecture that can process *any length input*



3. Recurrent Neural Networks (RNN)

A family of neural architectures

Core idea: Apply the same weights *W repeatedly*