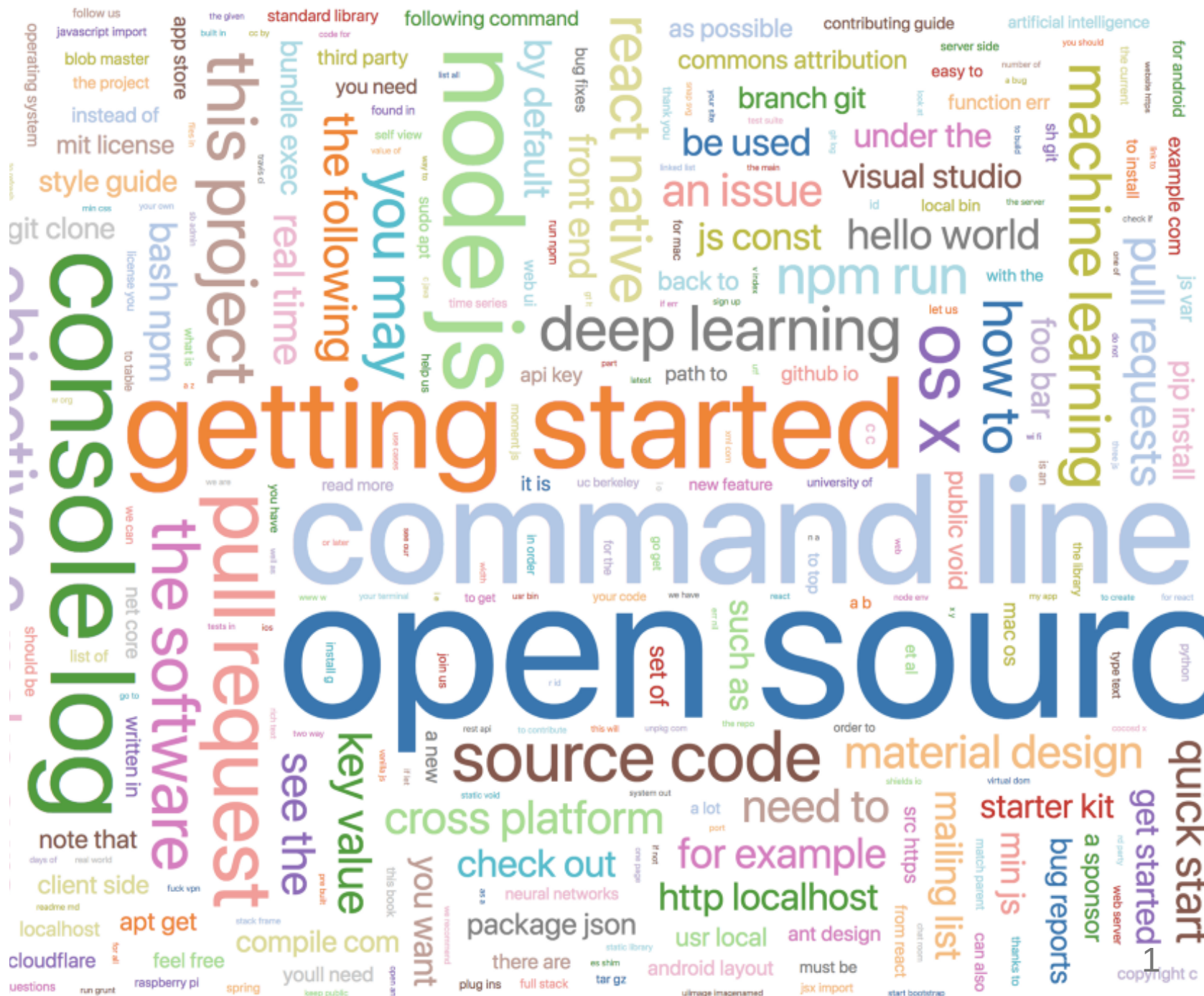


NLP course, 2022-23



# References

*Speech and Language Processing*. 3rd ed. draft Jan 2023. Dan Jurafsky, James H. Martin. <https://web.stanford.edu/~jurafsky/slp3/>.

- Chapter 3 : *N-gram language models*
- Chapter 7 : *Neural language models*, only sections 7.3, 7.5 and 7.6.1
- Appendix B : *Spelling correction and the noisy channel*

# Outline

1. Language model : what is and why do we want it
2.  $n$ -grams for LM
3. Problems and solutions
  - sparsity, unknown words
  - smoothing and backoff
4. A feed-forward neural LM
5. An application of  $n$ -grams: spelling correction

# Plan

- Day 1, 2h : theory (these slides)
- Homework : study and start solving the exercises
- Day 2, 2h : exercises, implement
  - text generation with a  $n$ -gram LM
  - with a neural LM
  - "real-words" spelling correction

## Text generation

<s> El →

El Girona Convention Bureau facilitarà , a través del bucle repetint successivament el mateix col·lapse que es fa una reivindicació al reconeixement social i l' evolució geològica

## "Real-words" spelling correction

Emma Woodhouse, **hand some**, clever, and rich, with a comfortable home **an** happy disposition, seemed to **unit** some of the best blessings of existence; and **hat** lived nearly twenty-one years in the world with very little **too** distress or vex her.

Emma Woodhouse, **handsome**, clever, and rich, with a comfortable home **and** happy disposition, seemed to **unite** some of the best blessings of existence; and **had** lived nearly twenty-one years in the world with very little **to** distress or vex her.

**Emma, Jane Austin**

# Language model: what and why

A **language model** is an algorithm / method / formula to assign a **probability** to a sentence or to a sequence of words, in a given natural language.

$$P(<s>Avui fa bon dia</s>) = 0.0123$$

$$P(<s>How's the weather like today?</s>) = 0.0034$$

Also, the *conditional* probability of a certain word given one or more previous words

$$P(\text{temps} \mid \text{Avui fa bon}) = 0.3$$

$$P(\text{like} \mid \text{How's the weather}) = 0.8$$

which means also the most probable word in the vocabulary  $V$  after a sequence

$$\arg \max_{w \in V} P(w \mid \text{How's the weather}) = \text{like}$$

# Applications

- Spell checking :

...How's the **wether** like...  $\rightarrow$

$$\arg \max_{w \in V} P(\text{How's the } w \text{ like}) = \text{weather}$$

- Speech recognition, we want

$$P(\text{recognize speech}) > P(\text{break a nice beach})$$



- Machine translation : Chinese  $\rightarrow$  English

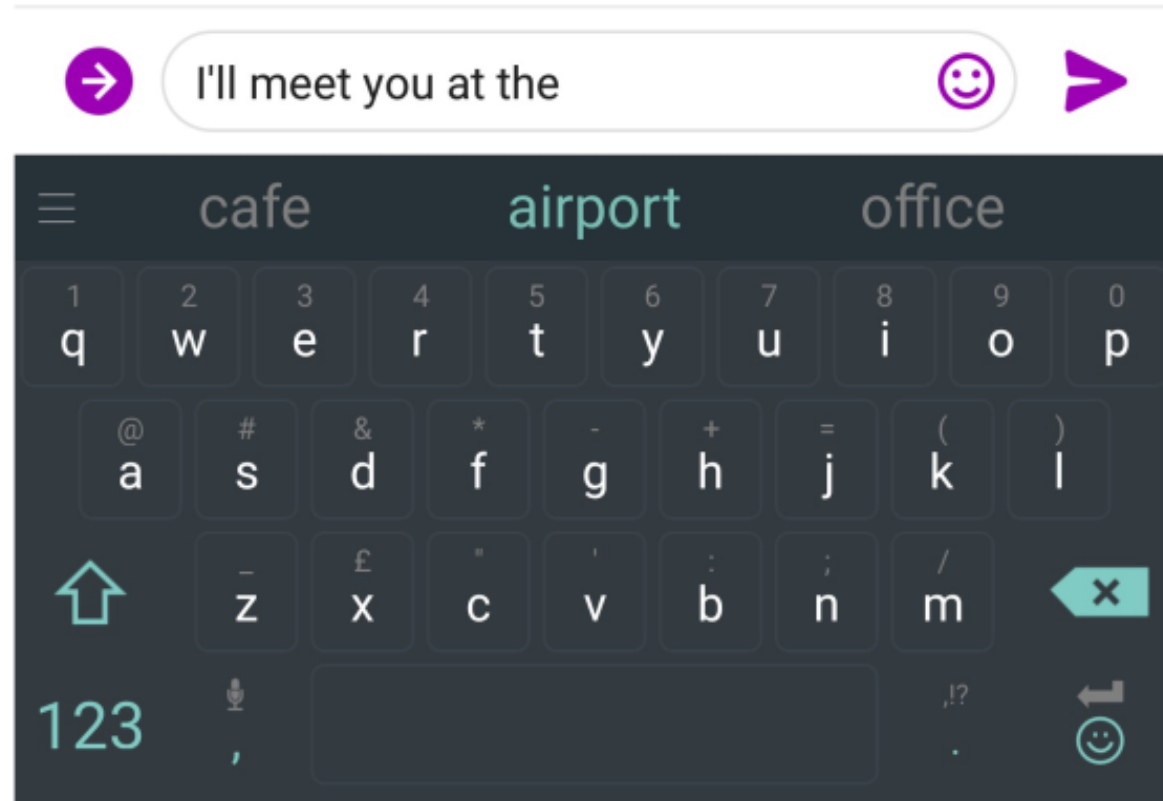
他 向 记者 介绍了 主要 内容

We want

$P(\text{he briefed reporters on the main contents of the statement}) >$

$P(\text{he introduced reporters to the main contents of the statement})$

- Word prediction



Sort  $w \in V$  by decreasing  $P(w \mid \text{meet you at the})$  and show top-3 words

- Text generation : given a sequence of starting words  $w_1 w_2 \dots w_k$ ,

$$w_1 w_2 \dots w_k \rightarrow w_1, w_2, \dots w_k w_{k+1} \dots w_n$$

$$w_i = \arg \max_{w \in V} P(w | w_{i-1}, w_{i-2} \dots w_{i-k}) , \quad i = k + 1 \dots n, \text{ or better}$$

$$w_i = \text{Sample} \left( w \sim P(w | w_{i-1}, w_{i-2} \dots w_{i-k}) \right)$$

$x \sim P(x)$  means random variable  $x$  follows distribution  $P(x)$

# Goal

**Let  $w_1 w_2 \dots w_k$  be a sequence of words. How to compute or estimate  $P(w_1 w_2 \dots w_k)$  and  $P(w_k | w_1 \dots w_{k-1})$  ?**

Naive solution:

1. take a large corpus
2. count occurrences of our sequence,  $C(w_1 w_2 \dots w_k)$
3. divide by total number of sequences of length  $n$  in the corpus

**Problems:**

- maybe  $w_1 w_2 \dots w_k$  doesn't exist in the corpus, almost surely if  $k$  large
- even if it exists, the corpus may not be large enough  $\rightarrow$  probability not reliable

**Solution:** approximate

By the chain rule of probability

$$\begin{aligned} P(w_1, w_2 \dots w_k) &= P(w_1) P(w_2|w_1) P(w_3|w_1, w_2) \dots P(w_k|w_1 \dots w_{k-1}) \\ &= \prod_{i=1}^k P(w_i|w_1 \dots w_{i-1}) \end{aligned}$$

Markov assumption: conditional probabilities depend only on the  $n + 1$  past words

$$P(w_k|w_1 \dots w_{k-1}) = P(w_k|w_{k-1} \dots w_{k-n+1})$$

If  $n = 2$ , only previous word matters:

$$P(w_1, w_2 \dots w_k) = P(w_1) P(w_2|w_1) P(w_3|w_2) \dots P(w_k|w_{k-1})$$

Do you think the approximation is reasonable ? Which is the best  $n$  ?

# $n$ -grams

*En una bella plana per on passava una bella aigua estaven gran res de besties selvatges qui volien elegir rei. Acord fo pres per la major part que el lleó fos rei. Més lo bou contrastava molt fortment a aquella elecció, i digué estes paraules*  
Llibre de les bèsties, Ramon Llull, 1289

Unigrams : En , una , bella , plana ... paraules

Bigrams : En una , una bella , bella plana ... digué estes , estes paraules

Trigrams : En una bella , una bella plana , ... digué estes paraules

A  $n$ -gram is a sequence of  $n$  consecutive words. Their frequencies will be useful to compute the probabilities of a LM.

Now you can compute the conditional  $\Rightarrow$  also the joint probability just by **counting n-grams**. For  $n = 2$ , **bigrams**

$$P(w_k|w_{k-1}) = \frac{C(w_{k-1}w_k)}{\sum_{w \in V} C(w_{k-1}w)} = \frac{C(w_{k-1}w_k)}{C(w_{k-1})}$$

$$P(w_{k-1}w_k) = P(w_k|w_{k-1})P(w_{k-1}) \quad , \quad P(w_k) = \frac{C(w_k)}{S}$$

$S$  is total number of words in the corpus.

$V$  set of *different* words in the corpus, or vocabulary

Why is this better ? It's much more probable than we can find all the  $w_{k-1}w_k$  in the corpus than  $w_1, \dots, w_n, n > 2$ .

Write the equations for  $n = 3$  and for the general case  $n \geq 2$

## A technicality

We need to augment the vocabulary  $V$  with special words

`<s>` = beginning of sentence

`</s>` = end of sentence

Now we can make a bigram with the first word of a sentence, and the set of all sentences has a proper probability distribution.

$$C(\mathbf{En} \mid \mathbf{<s>}) = 1$$

$$C(\mathbf{</s>} \mid \mathbf{rei}) = 2$$

$$P(\mathbf{Acord} \mid \mathbf{<s>}) = 1$$

Caution:  $C(w)$  is *not* the number of unigrams = occurrences of  $w$  but the number of times it's the first item of a bigram, including the two new words.



## Practical issues

- In practice it's better to use trigrams, 4-grams than bigrams. Why ?
- For trigrams you need to do like  $P(\text{restaurant} \mid \langle s \rangle \langle s \rangle)$  = probability that **restaurant** is first word of a sentence, etc.
- Because of possibility of underflow (large  $n$ , large corpus), instead of

$$P(w_2|w_1) P(w_3|w_2) \dots P(w_n|w_{n-1})$$

compute

$$\exp \left( \log P(w_2|w_1) + \log P(w_3|w_2) + \dots \log P(w_n|w_{n-1}) \right)$$

Compute  $P(\text{una bella})$  in the text above

$n$ -grams was one of the first takes of statistical NLP at building language models.

- 1948 first mention of  $n$ -grams by [Claude Shannon](#)
- 80s— 00s  $n$ -grams, rule-based systems for NLP
- 90s— recurrent NNs like [LSTM](#)
- 2003 Bengio et al. feed-forward neural LM
- 2010s **rise in computer power and data**
- 2012— computer vision taken by storm by deep learning → NLP
- 2017— attention models, transformer architectures which perform MUCH better

**towards**  
Read [this](#) data science [and this](#)



So, why study  $n$ -grams ?

- learn a *simple* approach to build a LM
- pre-deep learning era
- realize what were the problems that lead to *neural* LMs
- introduce LM applications
- we'll compare in the exercise a simple version of the two approaches
- and implement a type of spelling checker

# Problems and solutions

1. Unknown words
2. The  $n$  in  $n$ -grams and the "zeros problem"

# Unknown words

- The training corpus may be large but at test time there may always be new words
- How to compute probabilities then ?
- Replace the less frequent words in the vocabulary by the <UNK> word
- any word not in the training vocabulary is <UNK>
- estimate probabilities involving <UNK> considering it just like any other regular word

## The $n$

- the larger the context size  $n$  the better the  $n$ -grams LM is : more coherent sentences are generated
- but in the same domain of training corpus (eg. English in Shakespeare  $\neq$  English in Wall Street Journal)
- Shakespeare complete works corpus: 884,647 words, vocabulary 29,066 words

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.

The problem of large  $n$  is

- ratio of actual vs possible  $n$ -grams tends to zero :  $3e5$  found 4-grams vs  $V^4 = 7e17$  possible 4-grams
- It can not be but  $\rightarrow$  that | I | he | thou | so
- for most 4-grams there is only 1 continuation possible
- generated text is *literally* Shakespeare

Probabilities are not or less reliable : the zeros problem.



Recall : the idea of LM with  $n$ -grams is to compute probabilities just by **counting**. For  $n = 2$ , bigrams

$$P(w_k|w_{k-1}) = \frac{C(w_{k-1}w_k)}{\sum_{w \in V} C(w_{k-1}w)} = \frac{C(w_{k-1}w_k)}{C(w_{k-1})}$$

$$P(w_{k-1}w_k) = P(w_k|w_{k-1})P(w_{k-1}) \quad , \quad P(w_k) = \frac{C(w_k)}{S}$$

Probability computations suffer the **zeros problem**:

$$P(\text{can not be but simple}) = 0$$

in spite **simple** appears in (maybe many) other 5-grams

# Smoothing

Q: What to do with words in the vocabulary (not unknown) but appear in the test set in an unseen context ?

$$P(\text{simple} \mid \text{can not be but}) = 0$$

A: discount a bit of probability from frequent  $n$ -grams and give it to them

***add one*** or Laplace smoothing

$$P(w_k | w_{k-1}) = \frac{C(w_{k-1}w_k)}{C(w_{k-1})} \longrightarrow \frac{C(w_{k-1}w_k) + 1}{C(w_{k-1}) + |V|}$$

$|V|$  size of the vocabulary

## Backoff

**A different solution**, if we want  $P(w_k | w_{k-2}, w_{k-1})$  and there are no trigrams  $w_{k-2}w_{k-1}w_k$ , resort to  $P(w_k | w_{k-1})$ .

If there are not bigrams  $w_{k-1}w_k$ , resort to  $P(w_k)$

$P(w_k) > 0$  because  $w_k$  is a known word or <UNK>

## Stupid backoff

Little problem with backoff : the algorithm does not provide a proper probability distribution because there's no discount.

"Solution" : stupid backoff

$$S(w_k | w_{k-N+1} \dots w_{k-1}) = \begin{cases} \frac{C(w_{k-N+1} \dots w_{k-1} w_k)}{C(w_{k-N+1} \dots w_{k-1})} & \text{if numerator} > 0 \\ \lambda S(w_k | w_{k-N+1} \dots w_{k-2}) & \text{else} \end{cases}$$

$S$  and not  $P$  because again does not produce a probability distribution, but works better in practice (2007) for  $\lambda = 0.4$  in large corpora.

# Feed forward neural LM

*Feed-forward neural network :*

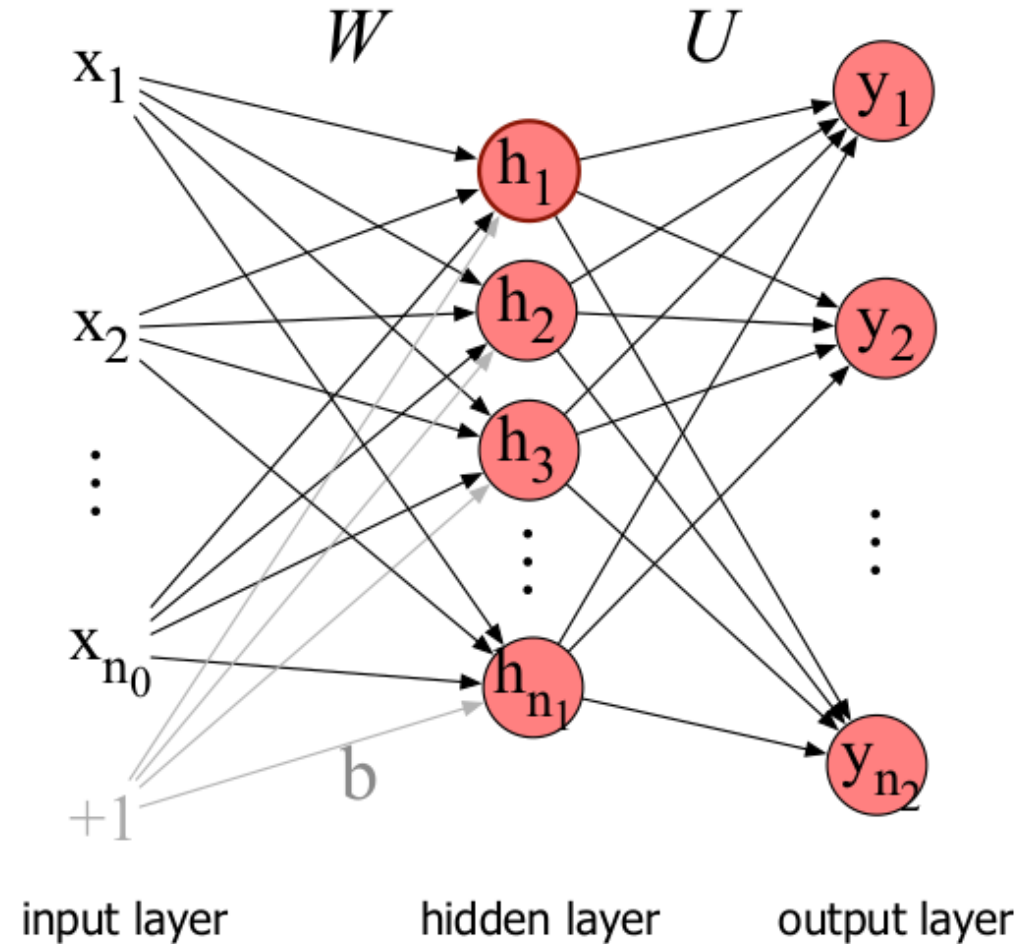
Connections between the nodes do not form a cycle. Information always moves one direction, never goes backwards.

A multilayer perceptron (MLP) is a type of FFNN.

$h_i = f(\mathbf{w} \cdot \mathbf{x} + b_i)$ ,  $f$  non-linearity like  $\sigma$ ,  $\tanh$ ,  $\text{ReLU}$

Equivalently,  $\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$

Result  $\mathbf{y} = \mathbf{U}\mathbf{h}$



Bengio et al.<sup>1</sup> introduced the first fee-forward NN language model in 2003.

### Advantages over $n$ -gram LM

- can handle much longer histories (length of network input)
- can generalize better over contexts of similar words
- more accurate at word-prediction.

However,

- more complex
- need to be trained (slow, energy)
- lack explainability of  $n$ -grams LM (counting, probs.)

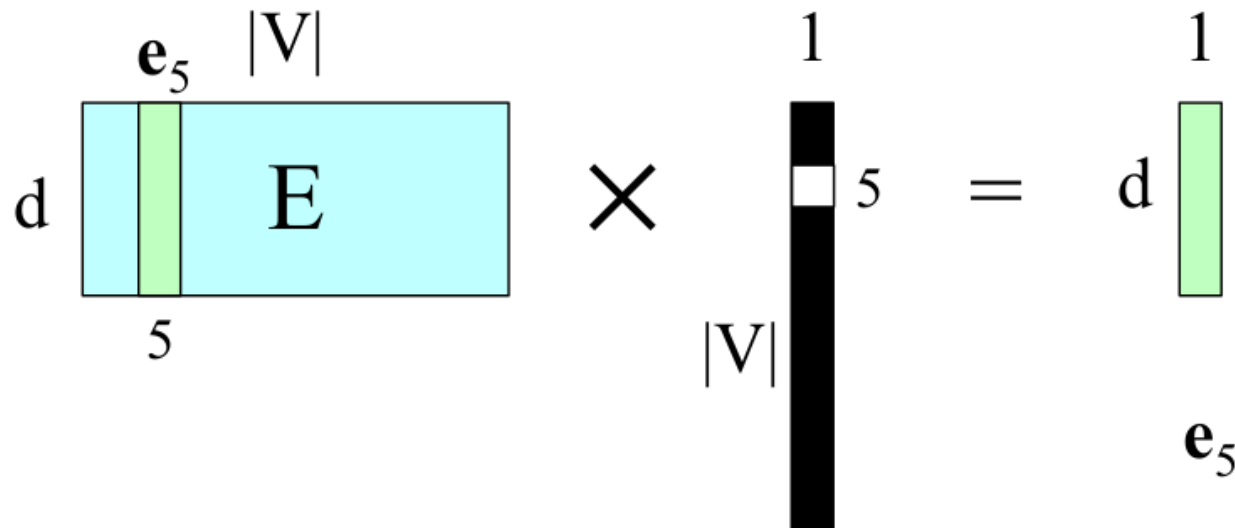
<sup>1</sup>[A neural probabilistic language model](#). Bengio, Y., R. Ducharme, P. Vincent, and C. Jauvin. *Journal of Machine Learning Research*, 2003.

## How's the feed-forward NN for language modeling ?

- input to NN is a sequence of words ['<s>', 'El', 'dia', 'que']
- each word is coded as a one-hot vector, equivalently, an id number

$$\text{El} \rightarrow w = [0, 0, 0, 0, 1, 0, \dots 0], w \in \{0, 1\}^{|V|}, w[5] = 1, \text{id} = 5$$

- first layer of NN changes the representation : from one-hot / id to a vector  $e \in \mathbb{R}^d$
- $e_5$  lives in a  $d$ -dimensional space, the **embedding E**





**The embedding is key:** it is initialized at random but later **learned** during training. It learns which words go with each other words in the training set = small Euclidean distance.

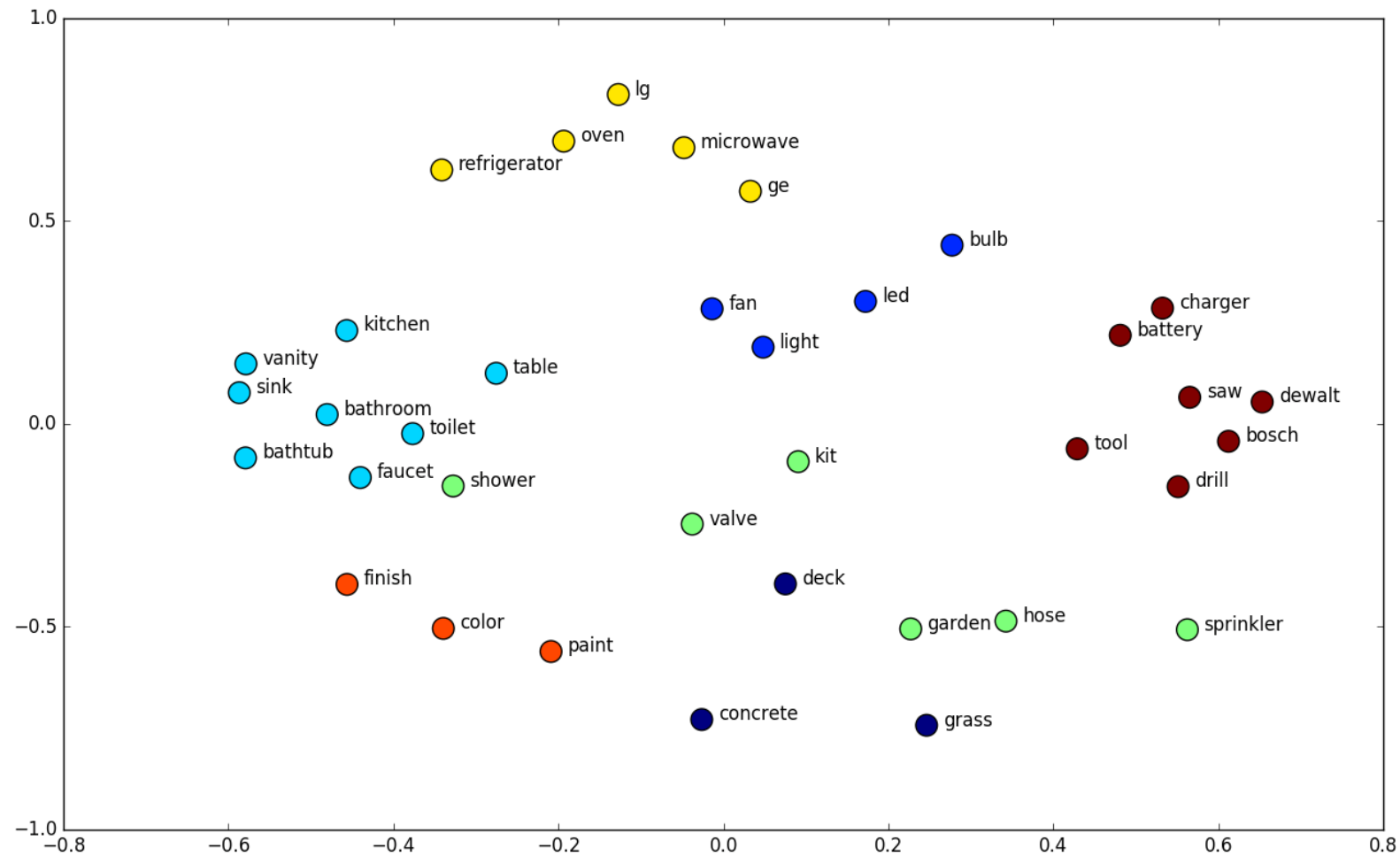
Using embeddings allows neural language models to generalize better to unseen data. For example, suppose we've seen this sentence in training:

I have to make sure that the cat gets fed

but have never seen the words “gets fed” after the word **dog**. Our test set has the prefix **I forgot to make sure that the dog gets**. What's the next word?

An n-gram language model will predict **fed** after **that the cat gets**, but not after **that the dog gets**. But a neural LM, **knowing that cat and dog have similar embeddings**, will be able to generalize from the **cat** context to **assign a high enough probability to fed even after seeing dog**.

Projection of  $d$ -dimensional embedding to 2d using t-SNE,



<https://neptune.ai/blog/word-embeddings-guide>

- the embeddings of the 4 input words are concatenated in a single vector  $x \in \mathbb{R}^{4d}$
- next comes a pair of fully connected layers

$$y_1 = \tanh(\mathbf{W}x + b_1)$$
$$y_2 = \mathbf{U}y_1 + x + b_2$$

- output scores  $y_2$  are passed through a softmax normalization to the probability  $p(w \mid [ \text{'<s>'}, \text{'El'}, \text{'dia'}, \text{'que'} ])$  for each word  $w \in V$

For details on how to **train** the network and the loss function (cross-entropy), see Sect. 7.6 of Jurafsky book and [exercise 2](#).

output distribution

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

hidden layer

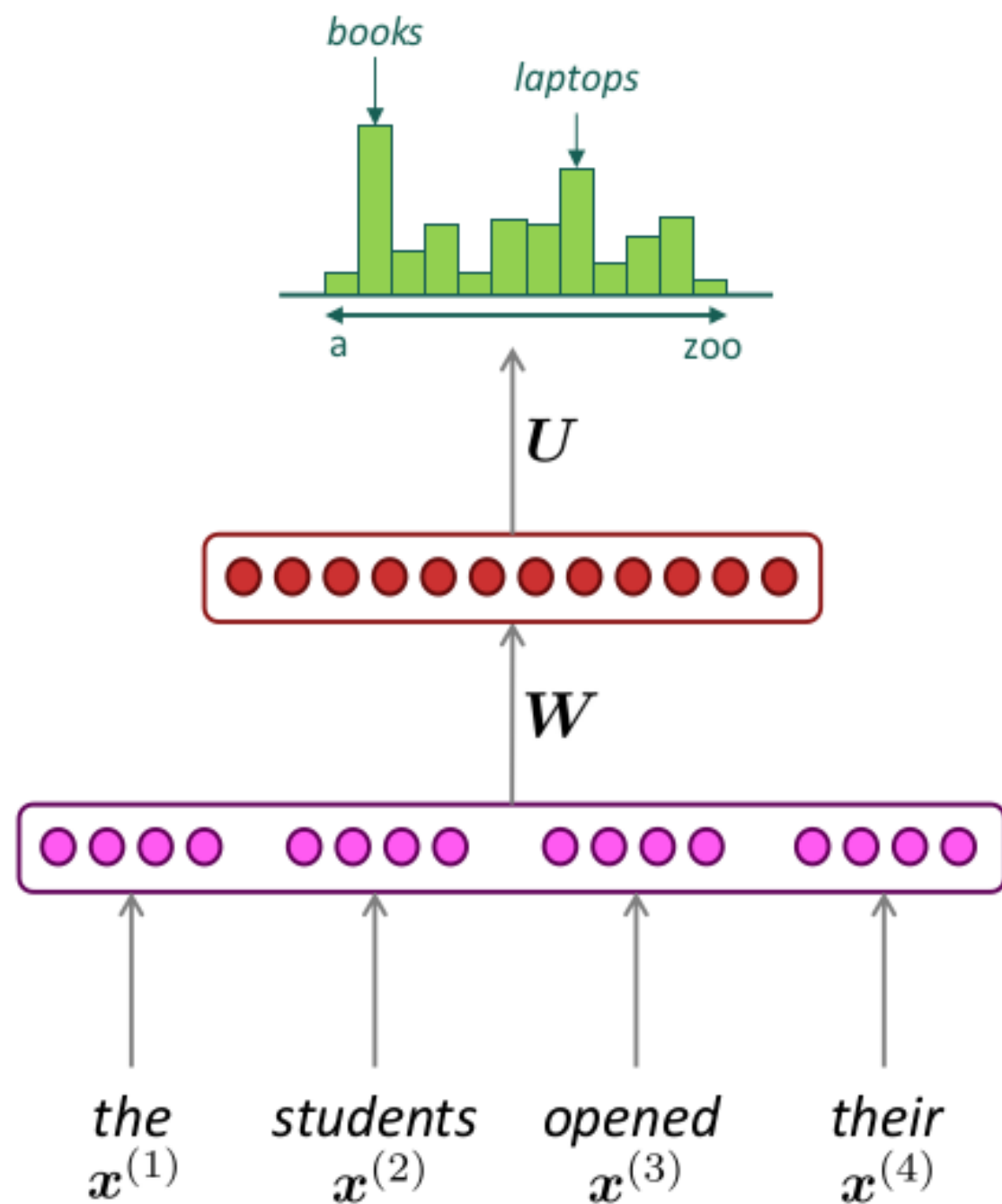
$$\mathbf{h} = f(\mathbf{W}\mathbf{e} + \mathbf{b}_1)$$

concatenated word embeddings

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

words / one-hot vectors

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



In Pytorch:

```
class NNLM(nn.Module):
    def __init__(self, num_classes, dim_input, dim_hidden, dim_embedding):
        super(NNLM, self).__init__()
        self.dim_input = dim_input
        self.dim_embedding = dim_embedding
        self.embeddings = nn.Embedding(num_classes, self.dim_embedding)
        self.hidden1 = nn.Linear(self.dim_input * self.dim_embedding,
                                   dim_hidden, bias=False)
        self.bias1 = nn.Parameter(torch.ones(dim_hidden)) # initialized to 1
        self.hidden2 = nn.Linear(dim_hidden, num_classes, bias=False)
        self.hidden3 = nn.Linear(self.dim_input * self.dim_embedding,
                                   num_classes, bias=False)
        self.bias2 = nn.Parameter(torch.ones(num_classes)) # initialized to 1

    def forward(self, X):
        word_embeds = self.embeddings(X)
        X = word_embeds.view(-1, self.dim_input * self.dim_embedding) # concatenate
        tanh = torch.tanh(self.bias1 + self.hidden1(X))
        output = self.bias2 + self.hidden3(X) + self.hidden2(tanh)
        return output
```

```

model = NNLM(num_classes=dataset.vocabulary_size, dim_input=4,
              dim_hidden=32, dim_embedding=50)
generated_words = ['<s>', 'El', 'dia', 'que']

while len(generated_words) < 100:
    input_ids = [dataset.word2id[w] for w in generated_words[-dim_input:]]
    # select last dim_input words

    pred = model(torch.tensor(input_ids).unsqueeze(0))
    probs = torch.nn.functional.softmax(pred, dim=1)
    output_id = torch.multinomial(probs, 1)
    # sample {0...num_classes-1} according to probabilities

    output_word = dataset.id2word[output_id.item()]
    generated_words += [output_word]

```

## Improvements over $n$ -gram LM:

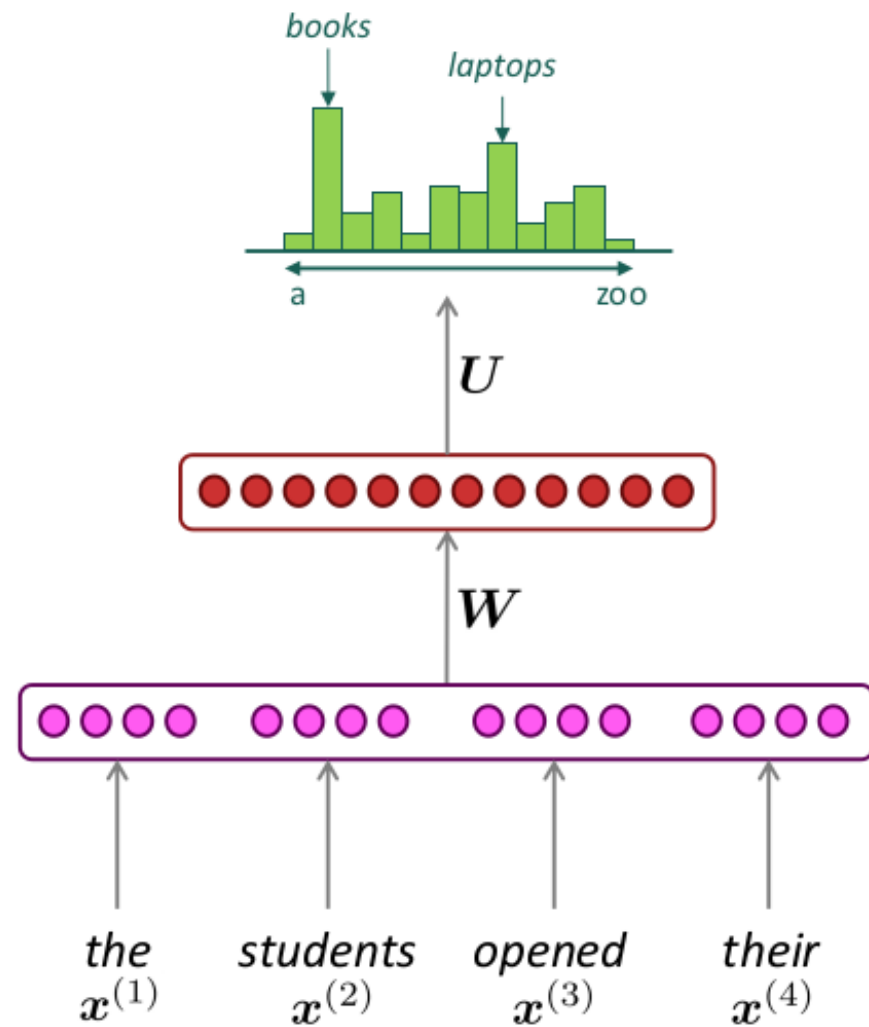
- No sparsity problem
- Don't need to store all observed  $n$ -grams

## 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  $W$ .

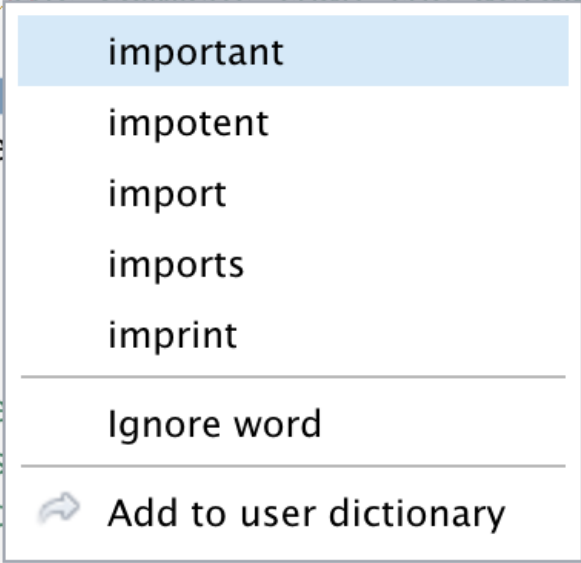
**No symmetry** in how the inputs are processed.

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



# Spell checking

```
1 #include <Rcpp.h>
2 using namespace Rcpp;
3
4 // Much more important comment than the default.
5 // I hope I don't
6 // [[Rcpp::export]]
7 NumericVector time
8   return x * 2;
9 }
10
11
12 // You can include
13 // (useful for tes
14 // run after the c
15 //
```



**impotent ?!**

Frequency of spelling errors vary

- from 1-2% for carefully retyping already printed text
- to 10-15% for web queries.



## Two types

- **Non-word spelling correction** : errors that result in non-words, not in the vocabulary

Emma Woodhouse, **hansome**, clever, and rich, with a **comfortable** home and happy disposition, seemed to unite some of the best **blesings** of existence; and had lived nearly twenty-one years in the world with very little to **distres** or vex her.

**Emma, Jane Austin**

- **Real word spelling correction** : spelling errors even if they accidentally result in an actual word

Emma Woodhouse, **hand some**, clever, and rich, with a comfortable home **an** happy disposition, seemed to **unit** some of the best blessings of existence; and **hat** lived nearly twenty-one years in the world with very little **too** distress or vex her.

**Emma, Jane Austin**

Strategy for non-word errors:

- find words  $w_1, w_2, \dots, w_k$  not in the vocabulary
- for each  $w_i$ 
  - find the words in the vocabulary at a Levenshtein<sup>1</sup> distance  $\leq 1$
  - **rank** these words by decreasing **probability of being the correct one**
  - show the list and let the user choose

<sup>1</sup> delete, add, replace 1 character, transpose 2 successive characters. **where**  $\rightarrow$  **were, there, here, wheer** . See Ch. 2, sect. 2.5 of Jurafsky's book.

## Strategy for real-word errors:

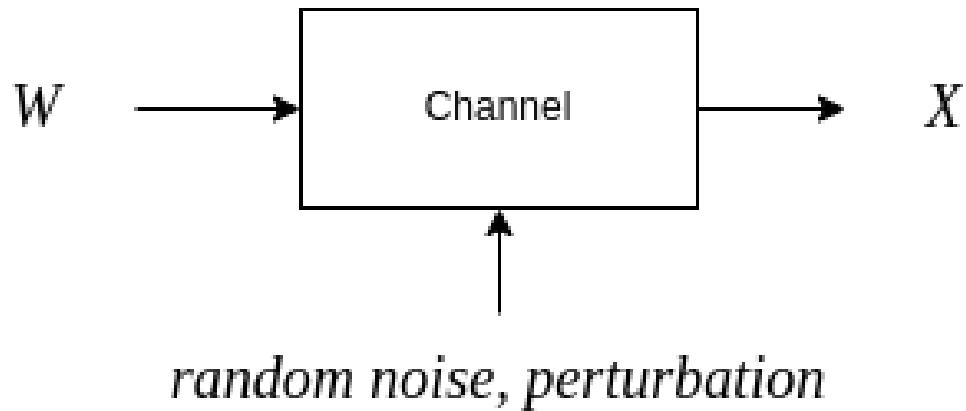
- more difficult because *any* word can be an error
- for each typed *sentence*  $X$ 
  - assume there is 0 or only 1 misspelled word
  - for each of its words find words in the vocabulary at distance  $\leq 1$
  - make all possible sentences in this way,  $C(X)$ , where  $X \in C(X)$
  - rank these sentences by decreasing **probability of being the correct one**
  - select  $W \in C(X)$  such that  $P(W|X) - P(X|X) \geq \text{threshold}$ ,  
(or better,  $\log P(W|X) - \log P(X|X) \geq \text{threshold}$ )
  - show list and let the user choose

$X = I$  wish you where here

$W \in C(X)$  and (posterior) probabilities :

I	wish	you	where	here	8.924506647995294e-12	
A	wish	you	where	here	1.0286322789468195e-17	
a	wish	you	where	here	1.4812615001448439e-18	
s	wish	you	where	here	2.547891284427092e-18	
If	wish	you	where	here	8.184815983017701e-18	
It	wish	you	where	here	3.262865831067868e-17	
I	fish	you	where	here	7.454921980723136e-20	
I	with	you	where	here	3.594682695244262e-16	
I	wish	You	where	here	3.392605965931279e-17	
I	wish	your	where	here	4.715633943530782e-15	
I	wish	you	here	here	3.8867450475411456e-14	
I	wish	you	were	here	1.903874150218112e-09	*****
I	wish	you	There	here	3.769850459645172e-14	
I	wish	you	Where	here	4.675783515838973e-15	
I	wish	you	there	here	2.818265257261675e-13	
I	wish	you	where	her	1.2012691689115732e-12	
I	wish	you	where	hers	1.0090459209673022e-14	
I	wish	you	where	were	2.9817306964583775e-13	
I	wish	you	where	There	6.508346190239099e-14	
I	wish	you	where	Where	8.072367367738417e-15	
I	wish	you	where	there	7.661465393231837e-12	
I	wish	you	where	where	3.582113019433922e-14	

How to compute the probabilities ? Using the **noisy-channel model** (at sentence level)



$W = w_1 w_2 \dots w_n$  correct sentence

$X = x_1 x_2 \dots x_n$  typed sentence (correct or no)

$C(X)$  set of candidate sentences to correct sentence =  $X$  plus sentences with 1 word changed to another real-word at Levenshtein distance 1

$$\textbf{Goal : } \hat{W} = \arg \max_{W \in C(X)} P(W|X)$$

We can not compute / model  $P(W|X)$  but by Bayes theorem

$$P(W|X) = \frac{P(X|W)P(W)}{P(X)}$$

$$\hat{W} = \arg \max_{W \in C(X)} P(W|X) = \arg \max_{W \in C(X)} P(X|W)P(W)$$

- $P(X|W)$  likelihood, **channel model**,  
how probable is that  $W$  is transformed into  $X$
- $P(W)$  prior
- $P(X)$  evidence, doesn't matter because we optimize with respect to  $W$

Appendix: Bayes theorem

How to compute  $P(W)$  ? Using a  $n$ -gram language model

- $W$  is a sentence
- append  $(n - 1)$   $\langle s \rangle$  at the beginning and one  $\langle /s \rangle$  at the end
- $W = w_1 w_2 \dots w_k$
- by the Markovianity assumption
  - bigrams  $P(w_1, w_2 \dots w_k) = P(w_2|w_1) P(w_3|w_2) \dots P(w_k|w_{k-1})$
  - trigrams  $P(w_1, w_2 \dots w_k) = P(w_3|w_1 w_2) \dots P(w_k|w_{k-2} w_{k-1})$
  - ...

But a neural language model would do also, if the output is passed to a softmax layer.



How to compute  $P(X|W)$  ?

- $X$  sentence  $x_1 x_2 \dots x_k$  with same number of words than  $W = w_1 w_2 \dots w_k$
- $W$  is each sentence in  $C(X)$
- $P(X|W) = \prod_{i=1}^k p(x_i|w_i)$
- $C(x)$  set of words in the vocabulary at a distance  $\leq 1$  of  $x$ , thus including  $x$  itself
- A simplification : the probability of writing  $x_i$  instead of  $w_i$ ,  $i = 1 \dots k$  is

$$p(x|w) = \begin{cases} \alpha & \text{if } w = x, \text{ for instance } 0.95 \\ \frac{1 - \alpha}{|C(x)| - 1} & \text{if } w \in C(x) \text{ but } x \neq w \\ 0 & \text{else} \end{cases}$$

# Exercise 1 : Sample a $n$ -gram LM

1. Download a corpus in Catalan, Spanish or English
2. Process to extract tokens, sentences
3. Create a LM from its bigrams, trigrams, 4-grams
4. Sample the LM to generate text

NLTK has support for this and is well [documented](#).

Link to [Google Colab notebook](#)

## Exercise 2

Compare a  $n$ -grams LM with the feed-forward LM

1. Load the `cess_cat` corpus, already tokenized and segmented in sentences.
2. Add `<s>` , `</s>`
3. Make a Pytorch `Dataset` that returns batches of  $n$ -grams
4. Build the FF neural network (code provided)
5. Train it (GPU in Google Colab)
6. Generate text and compare
7. Optionally, visualize the learned word embedding

Link to [Google Colab notebook](#)

## Exercise 3

Build a spell corrector based on a LM. For the sake of simplicity we'll only consider the *real word* part.

The goal is, given a text, suggest a *ranked* list of words for each potentially misspelled word that still is a real word:

I wish you **where** here

To rank the list of real words close to the misspelled one, use the probabilities of the  $n$ -gram and / or neural LM and Bayes.

Link to [Google Colab notebook](#)

# Recap

- a LM is a way to compute the **probability of a sequence of words**, or the **conditional probability of a word** given preceeding words
- useful for a number of **applications**
- with **n-grams** we can easily build a simple LM
- the choice of  $n$  matters
- **zeros problem** solved with **Laplace smoothing** or **backoff**
- can also build a LM with a simple **feed-forward neural network**, avoiding the zeros problem
- these LMs can not compete with present day LMs like GPT, they are 20 years older!
- but simple and still useful to implement a spell checker

# Appendix: conditional probabilities and Bayes

A probability space is  $(\Omega, \mathcal{A}, P)$  with

- $\Omega$  sampling space, set of possible outcomes of a random experiment
- $\mathcal{A}$   $\sigma$ -algebra on  $\Omega$ , for instance set of possible subsets of  $\Omega$ ,  $\mathcal{P}(\Omega)$  "parts of  $\Omega$ "
- $P$  function that assigns a probability measure in  $[0, 1]$  to an element of  $\mathcal{A}$

A (discrete) random variable  $X$  is an application  $X : \Omega \rightarrow \mathbb{R}$  so that we can write something like  $P(X \leq a)$ ,  $P(X = b)$ ,  $P(a < X \leq b)$ ,  $a, b \in \mathbb{R}$



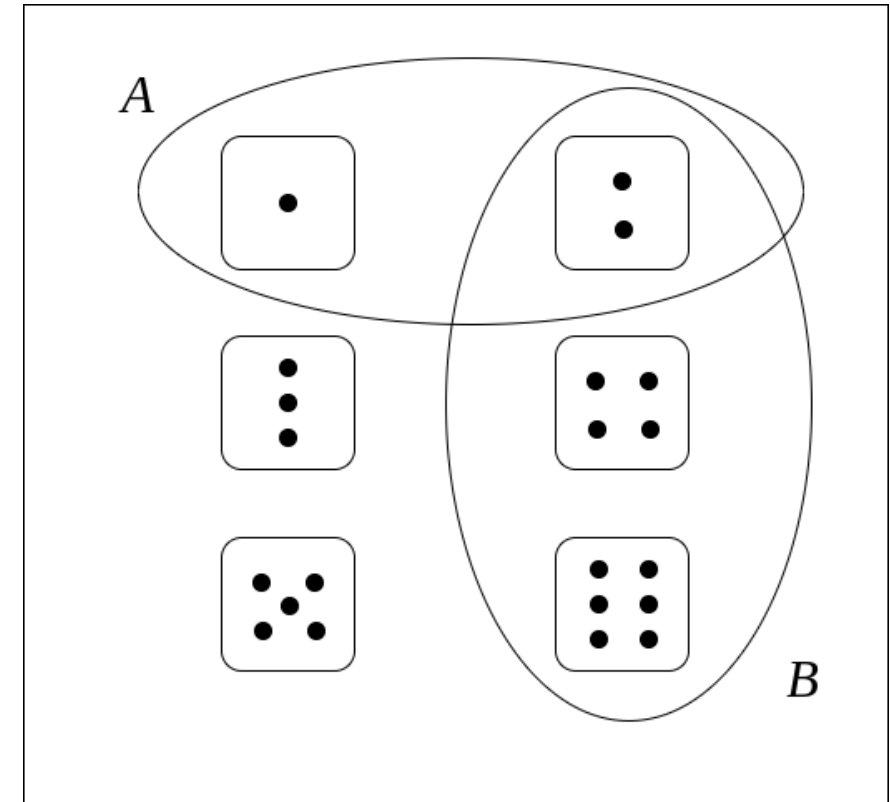
- experiment : throw a dice *once*
- $\Omega = \{1, 2, 3, 4, 5, 6\}$
- $\mathcal{A} = \{\emptyset, \{1\}, \dots \{6\}, \{1, 2\}, \dots \{5, 6\}, \{1, 2, 3\}, \dots \Omega\}$
- $P(\{3\}) = 1/6, P(\{1, 2, 3\}) = 1/2, P(\emptyset) = 0, P(\Omega) = 1 \dots$
- $X$  is Euros you win or loose :  $X(\{1\}) = -3, X(\{2\}) = X(\{4\}) = 1,$   
 $X(\{3\}) = X(\{5\}) = -1, X(\{6\}) = 2,$   
what is  $P(X > 1.5)$  ?

**Conditional probability** :  $A, B \in \mathcal{A}$   
(events), if  $P(B) > 0$ , then

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Meaning: probability of  $A$  if we know  
 $B$  has occurred or is true.

$$P(1 \text{ or } 2 \mid \text{outcome is even}) = \frac{1/6}{1/2} = 1/3$$





**Bayes theorem** or conditioning reversal:

$$P(B|A) = \frac{P(A|B) P(B)}{P(A)}$$

and similarly with random variables  $X, Y$ .

Example 2.3. Jo has a test for a nasty disease. We denote Jo's state of health by the variable  $a$  and the test result by  $b$ .

$$\begin{array}{ll} a = 1 & \text{Jo has the disease} \\ a = 0 & \text{Jo does not have the disease.} \end{array} \quad (2.12)$$

The result of the test is either 'positive' ( $b = 1$ ) or 'negative' ( $b = 0$ ); the test is 95% reliable: in 95% of cases of people who really have the disease, a positive result is returned, and in 95% of cases of people who do not have the disease, a negative result is obtained. The final piece of background information is that 1% of people of Jo's age and background have the disease.

OK – Jo has the test, and the result is positive. What is the probability that Jo has the disease?

**Solution.** We write down all the provided probabilities. The test reliability specifies the conditional probability of  $b$  given  $a$ :

$$\begin{aligned} P(b=1 \mid a=1) &= 0.95 & P(b=1 \mid a=0) &= 0.05 \\ P(b=0 \mid a=1) &= 0.05 & P(b=0 \mid a=0) &= 0.95; \end{aligned} \quad (2.13)$$

and the disease prevalence tells us about the marginal probability of  $a$ :

$$P(a=1) = 0.01 \quad P(a=0) = 0.99. \quad (2.14)$$

From the marginal  $P(a)$  and the conditional probability  $P(b \mid a)$  we can deduce the joint probability  $P(a, b) = P(a)P(b \mid a)$  and any other probabilities we are interested in. For example, by the sum rule, the marginal probability of  $b=1$  – the probability of getting a positive result – is

$$P(b=1) = P(b=1 \mid a=1)P(a=1) + P(b=1 \mid a=0)P(a=0). \quad (2.15)$$

Jo has received a positive result  $b=1$  and is interested in how plausible it is that she has the disease (i.e., that  $a=1$ ). The man in the street might be duped by the statement ‘the test is 95% reliable, so Jo’s positive result implies that there is a 95% chance that Jo has the disease’, but this is incorrect. The correct solution to an inference problem is found using Bayes’ theorem.

$$P(a=1 | b=1) = \frac{P(b=1 | a=1)P(a=1)}{P(b=1 | a=1)P(a=1) + P(b=1 | a=0)P(a=0)} \quad (2.16)$$

$$= \frac{0.95 \times 0.01}{0.95 \times 0.01 + 0.05 \times 0.99} \quad (2.17)$$

$$= 0.16. \quad (2.18)$$

So in spite of the positive result, the probability that Jo has the disease is only 16%. □