

# Introduction & Vector Representations of Text

## COM6513 Natural Language Processing

Nikos Aletras

Computer Science Department

Week 1  
Spring 2022



The  
University  
Of  
Sheffield.

## Part I: Introduction

# Course Practicalities

Lecture slides & lab assignments:

- Blacboard

Lab demonstrators:

- Danae Sanchez Villegas
- Tomas Goldsack
- Huiyin Xue
- Ahmed Alajrami
- Mali Jin
- Hongrui Shi

# Course Practicalities

Google Group for announcements and assignments QA ([join using @sheffield.ac.uk](#))

- <https://groups.google.com/a/sheffield.ac.uk/g/com6513---nlp-2022-group>

# Assessment

- **50% exam** where **everything** is assessed: lecture slides, bibliographical references, classroom discussion, lab content, etc.
- **20% Assignment 1, 30% Assignment 2:**
  - Deadlines TBA
  - Do them (so that we can help) and please do not attempt to plagiarise!

# Feedback

To you:

- During the lab sessions on assignments
- Questions on the Google group

# Feedback

To you:

- During the lab sessions on assignments
- Questions on the Google group

And to me:

- NSS evaluation
- Module evaluation

# Feedback

To you:

- During the lab sessions on assignments
- Questions on the Google group

And to me:

- NSS evaluation
- Module evaluation
- Some changes from last year were student suggestions.

# Course goals

- Learn how to develop systems to perform natural language processing (NLP) tasks
- Understand the main machine learning (ML) algorithms for learning such systems from data
- Become familiar with important NLP applications
- Have knowledge of state-of-the art NLP and ML methods

# Prerequisites

## Essential

- Text Processing ([COM3110/4115/6115](#))
- Machine Learning and Adaptive Intelligence ([COM4509/6509](#))
- Programming skills (i.e. Python).

# Prerequisites

## Essential

- Text Processing ([COM3110/4115/6115](#))
- Machine Learning and Adaptive Intelligence ([COM4509/6509](#))
- Programming skills (i.e. Python).

## Optional (but strongly recommended)

- Basic knowledge in Linguistics: see this [tutorial](#) by Emily Bender

# Why Natural Language Processing?

# Why Natural Language Processing?

The screenshot shows the Google Translate interface. At the top, there are language selection dropdowns for English, Spanish, French, and English - detected. Below these, the input text "Hello World" is shown in English, and the translated text "Hola Mundo" is shown in Spanish. There are edit icons and a "Suggest an edit" button at the bottom right of the translation box. On the left side, there is a "See also" section with a link to "World". At the bottom, there is a navigation bar with links for "Google Translate for Business", "Translator Toolkit", "Website Translator", "Global Market Finder", "Turn off instant translation", "About Google Translate", "Mobile", "Community", "Privacy & Terms", "Help", and "Send feedback".

## Machine Translation

# Why Natural Language Processing?



Question Answering

# Why Natural Language Processing?



Playing Jeopardy

# Why Natural Language Processing?

Facebook needs a "Do some fact checking before posting" button. It would spare a lot of people looking like an idiot!

som~~e~~ecards  
user card



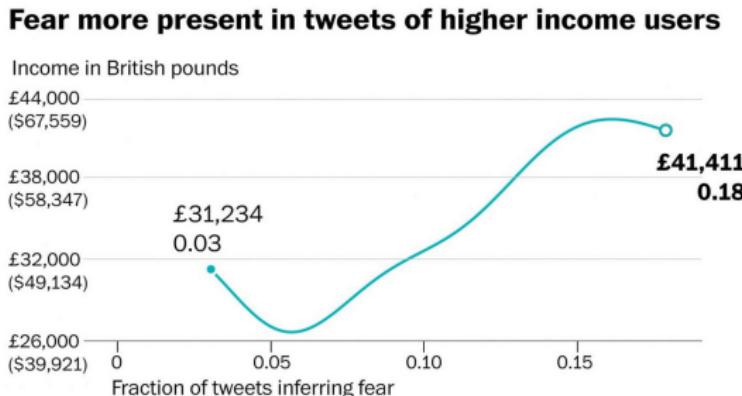
Fact Checking

# Why Natural Language Processing?

The screenshot shows the top navigation bar of The Guardian website. It features a yellow 'Support The Guardian' section with 'Contribute →' and 'Subscribe →' buttons. To the right is a 'Sign in' button and the 'The Guardian' logo. Below the main header is a secondary navigation bar with categories: News, Opinion, Sport, Culture, and Lifestyle. A menu icon (three horizontal lines) is also present. The main content area displays a news article titled 'Artificial intelligence (AI)' followed by the headline 'Artificial intelligence 'judge' developed by UCL computer scientists'. A subtext below the headline reads: 'Software program can weigh up legal evidence and moral questions of right and wrong to predict the outcome of trials'.

Assist in Human Decision Making

# Why Natural Language Processing?



Source: University of Pennsylvania research article "Studying User Income through Language, Behaviour and Affect in Social Media" by Daniel Preoțiuc-Pietro, Svitlana Volkova, Vasileios Lampos, Yoram Bachrach and Nikolaos Aletras

THE WASHINGTON POST

Computational Social Science

# Why is NLP challenging?

# Why is NLP challenging?

- Natural languages (unlike programming languages) are not designed; they **evolve!**
  - new words appear constantly
  - syntactic rules are flexible
  - ambiguity is inherent

# Why is NLP challenging?

- Natural languages (unlike programming languages) are not designed; they **evolve!**
  - new words appear constantly
  - syntactic rules are flexible
  - ambiguity is inherent
- world knowledge is necessary for interpretation

# Why is NLP challenging?

- Natural languages (unlike programming languages) are not designed; they **evolve!**
  - new words appear constantly
  - syntactic rules are flexible
  - ambiguity is inherent
- world knowledge is necessary for interpretation
- many languages, dialects, styles, etc.

# Why statistical NLP?

- Traditional rule-based artificial intelligence (symbolic AI):
  - requires expert knowledge to engineer the rules
  - not flexible to adapt in multiple languages, domains, applications
- Learning from data (machine learning) adapts:
  - to evolution: just learn from new data
  - to different applications: just learn with the appropriate target representation

## Words of caution

- When exploring a task, it is often useful to experiment with some simple rules to test our assumptions
- In fact, for some tasks rule-based approaches rule, especially in industry:
  - question answering
  - natural language generation

## Words of caution

- When exploring a task, it is often useful to experiment with some simple rules to test our assumptions
- In fact, for some tasks rule-based approaches rule, especially in industry:
  - question answering
  - natural language generation
- If we don't know how to perform a task, it is unlikely that a ML algorithm will find it out for us

# NLP =? ML

- NLP is a confluence of computer science, artificial intelligence (AI) and linguistics
- ML provides statistical techniques for problem solving by learning from data (current dominant AI paradigm)
- ML is **often** used in modelling NLP tasks

# NLP =? Computational Linguistics

- Both mostly use text as data
- In Computational Linguistics (CL), computational/statistical methods are used to support the study of linguistic phenomena and theories
- In NLP, the scope is more general. Computational methods are used for translating text, extracting information, answering questions etc.

# NLP =? Computational Linguistics

- Both mostly use text as data
- In Computational Linguistics (CL), computational/statistical methods are used to support the study of linguistic phenomena and theories
- In NLP, the scope is more general. Computational methods are used for translating text, extracting information, answering questions etc.

The top NLP scientific conference is called:

Annual Meeting of the Association for Computational Linguistics (ACL)

## Other Related fields

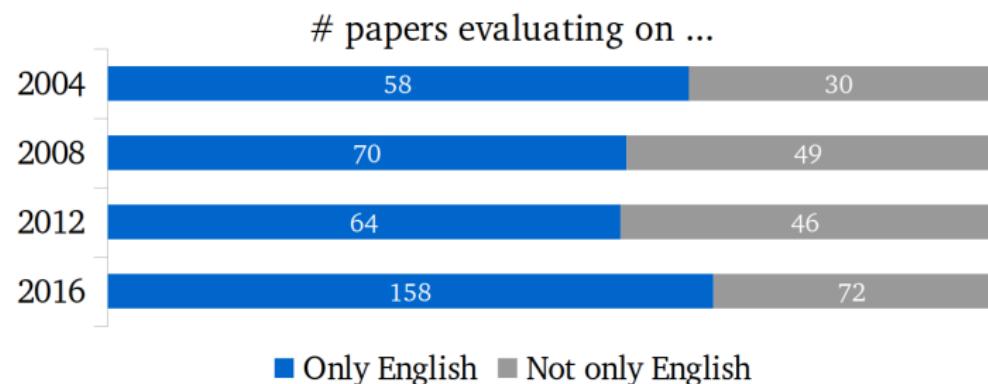
- Speech Processing
- Machine Learning
- Artificial Intelligence
- Search & Information Retrieval
- Statistics
- Any field that involves processing language:
  - literature, history, etc. (i.e. digital humanities)
  - biology
  - social sciences (sociology, psychology, law)

## Some food for thought

- 6,000 languages in the world, but in research papers?

# Some food for thought

- 6,000 languages in the world, but in research papers?



<http://sjmielke.com/acl-language-diversity.htm>

- NLP research has an English bias, our work is cut out!

# Course overview

- **Lecture 1:** Introduction and Vector Representations of Text
- **Lecture 2:** Text Classification with Logistic Regression
- **Lecture 3:** Statistical Language Modelling
- **Lecture 4:** Sequence Labelling and Part-of-Speech Tagging
- **Lecture 5:** Dependency Parsing

## Course overview (cont.)

- **Lecture 6:** Feed-forward Networks: Revisiting Text Classification and Word Vectors
- **Lecture 7:** Recurrent Networks and Neural Language Modelling
- **Lecture 8:** Information Extraction (Guest Lecture by Dr. Samuel Mensah)
- **Lecture 9:** Neural Seq2Seq Models for Machine Translation (Guest Lecture by Dr. Harish Tayyar Madabushi)
- **Lecture 10:** Transfer Learning for NLP

# Course Bibliography

- Jurafsky and Martin. 2008. Speech and Language Processing, Prentice Hall [3rd edition]
- Christopher D. Manning and Hinrich Schütze. 1999. Foundations of Statistical Natural Language Processing, MIT Press.
- Yoav Goldberg. 2017. Neural Network Methods in Natural Language Processing (Synthesis Lectures on Human Language Technologies), Morgan & Claypool Publishers, [A Primer on Neural Networks for NLP]
- Jacob Eisenstein. 2019. Introduction to Natural Language Processing. MIT Press. (A draft can be found [here](#))
- other materials referenced at the end of each lecture

# Opinion Poll time!

How long do you think it will take us to develop NLP systems that understand human language and have intelligence similar to humans?

Cast your vote here: <https://forms.gle/B6up6zm9MjiYakMg6>

## Part II: Vector Representations of Text

- Why do we need vector representations of text?
- How can we transform a raw text to a vector?

# Vectors and Vector Spaces

- A vector (i.e. embedding)  $\mathbf{x}$  is a **one-dimensional array** of  $d$  elements (coordinates), that can be identified by an index  $i \in d$ . e.g.  $x_1 = 0$

x	2	0	...	5
index	1	2	...	d

# Vectors and Vector Spaces

- A vector (i.e. embedding)  $x$  is a **one-dimensional array** of  $d$  elements (coordinates), that can be identified by an index  $i \in d$ . e.g.  $x_1 = 0$

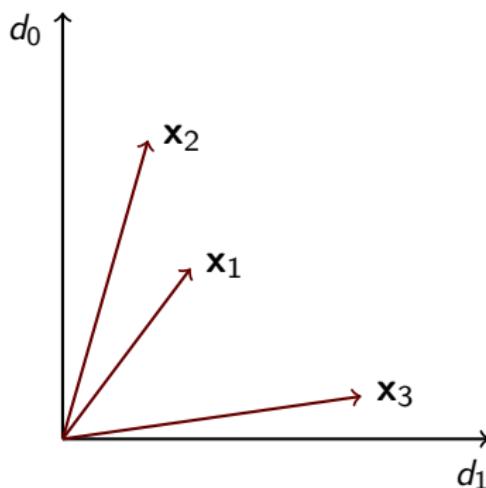
$x$	2	0	...	5
index	1	2	...	d

- A collection of  $n$  vectors is a **matrix  $X$**  with size  $n \times d$  - also called a **vector space**. e.g.  $X[2, 1] = -2$

$n \left\{ \begin{array}{|c|c|c|c|} \hline & 2 & 0 & \dots & 5 \\ \hline & -2 & 9 & \dots & 0 \\ \hline & & & \dots & \\ \hline & 0 & 2 & \dots & 0 \\ \hline \end{array} \right. \underbrace{\phantom{000}}_{d}$

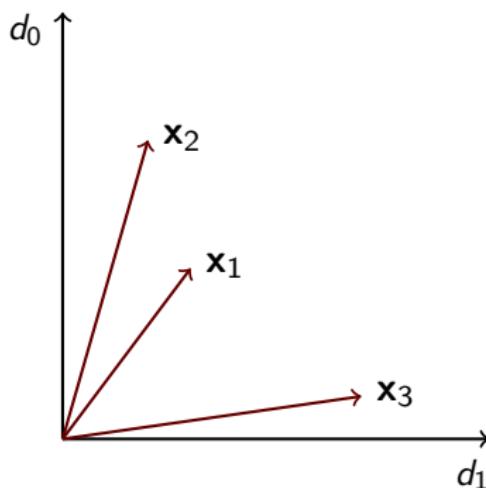
- Note that in Python indices start from 0!

## Example of a vector space



$d_i$  ( $i \in 1, 2$ ) are the coordinates and  $x_j$  are vectors

## Example of a vector space



$d_i$  ( $i \in 1, 2$ ) are the coordinates and  $x_j$  are vectors

How can we measure that  $x_1$  is closer to  $x_2$  than to  $x_3$ ?

# Vector Similarity

- **Dot (inner) product:** takes two equal-length sequences of numbers (i.e. vectors) and returns a single number.

$$\text{dot}(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \cdot \mathbf{x}_2 = \mathbf{x}_1 \mathbf{x}_2^\top = \sum_{i=1}^d x_{1,i} x_{2,i} = \\ x_{1,1} x_{2,1} + \dots + x_{1,d} x_{2,d}$$

# Vector Similarity

- **Dot (inner) product:** takes two equal-length sequences of numbers (i.e. vectors) and returns a single number.

$$\text{dot}(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \cdot \mathbf{x}_2 = \mathbf{x}_1 \mathbf{x}_2^\top = \sum_{i=1}^d x_{1,i} x_{2,i} = \\ x_{1,1} x_{2,1} + \dots + x_{1,d} x_{2,d}$$

- **Cosine similarity:** normalise dot product ([0, 1]) by dividing with vectors' lengths (or magnitude or norm)  $|\mathbf{x}|$ .

$$\text{cosine}(\mathbf{x}_1, \mathbf{x}_2) = \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{|\mathbf{x}_1| |\mathbf{x}_2|} = \frac{\sum_{i=1}^d x_{1,i} x_{2,i}}{\sqrt{\sum_{i=1}^d (x_{1,i})^2} \sqrt{\sum_{i=1}^d (x_{2,i})^2}}$$

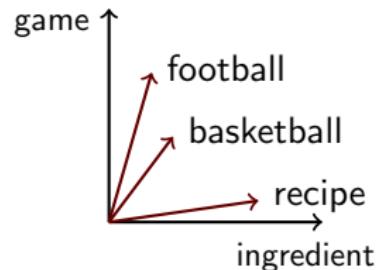
$$|\mathbf{x}| = \sqrt{\mathbf{x} \cdot \mathbf{x}} = \sqrt{x_1^2 + \dots + x_d^2}$$

## Vector Spaces of Text

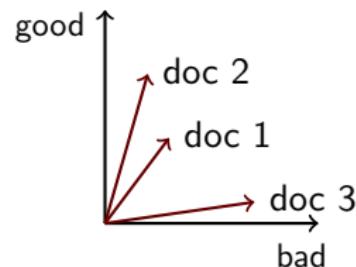
- Any ideas what are rows and columns for text data?

# Vector Spaces of Text

word-word (term-context)



document-word (bag-of-words)



# Why do we need vector representations of text?

- Encode the meaning of words so we can compute **semantic similarity** between them. E.g. is basketball more similar to football or recipe?

# Why do we need vector representations of text?

- Encode the meaning of words so we can compute **semantic similarity** between them. E.g. is basketball more similar to football or recipe?
- **Document retrieval**, e.g. retrieve documents relevant to a query (web search)

# Why do we need vector representations of text?

- Encode the meaning of words so we can compute **semantic similarity** between them. E.g. is basketball more similar to football or recipe?
- **Document retrieval**, e.g. retrieve documents relevant to a query (web search)
- Apply **Machine Learning** on textual data, e.g. clustering/classification algorithms operate on vectors. **We are going to see a lot of this during this course!**

# Text units

## Raw text:

*As far as I'm concerned, this is Lynch at his best. 'Lost Highway' is a dark, violent, surreal, beautiful, hallucinatory masterpiece. 10 out of 10 stars.*

# Text units

Raw text:

*As far as I'm concerned, this is Lynch at his best. 'Lost Highway' is a dark, violent, surreal, beautiful, hallucinatory masterpiece. 10 out of 10 stars.*

- **word (token/term)**: a sequence of one or more characters excluding whitespaces. Sometimes it consists of n-grams.

# Text units

## Raw text:

*As far as I'm concerned, this is Lynch at his best. 'Lost Highway' is a dark, violent, surreal, beautiful, hallucinatory masterpiece. 10 out of 10 stars.*

- **word (token/term)**: a sequence of one or more characters excluding whitespaces. Sometimes it consists of n-grams.
- **document (text sequence/snippet)**: sentence, paragraph, section, chapter, entire document, search query, social media post, transcribed utterance, pseudo-documents (e.g. all tweets posted by a user), etc.

# Text units

## Raw text:

*As far as I'm concerned, this is Lynch at his best. 'Lost Highway' is a dark, violent, surreal, beautiful, hallucinatory masterpiece. 10 out of 10 stars.*

- **word (token/term)**: a sequence of one or more characters excluding whitespaces. Sometimes it consists of n-grams.
- **document (text sequence/snippet)**: sentence, paragraph, section, chapter, entire document, search query, social media post, transcribed utterance, pseudo-documents (e.g. all tweets posted by a user), etc.
- How can we go from **raw** text to a **vector**?

# Text Processing: Tokenisation

**Tokenisation** to obtain tokens from raw text. Simplest form: **split text on whitespaces** or use **regular expressions**.

Raw text:

*As far as I'm concerned, this is Lynch at his best. 'Lost Highway' is a dark, violent, surreal, beautiful, hallucinatory masterpiece. 10 out of 10 stars.*

Tokenised text:

*As far as I'm concerned , this is Lynch at his best . ' Lost Highway ' is a dark , violent , surreal , beautiful , hallucinatory masterpiece . 10 out of 10 stars .*

## Text Processing: Other pre-processing options

- Other pre-processing steps may follow: lowercasing, punctuation/number/stop/infrequent word removal and stemming (remember COM4115/6115)

### Tokenised text:

*As far as I'm concerned , this is Lynch at his best . ' Lost Highway ' is a dark , violent , surreal , beautiful , hallucinatory masterpiece . 10 out of 10 stars .*

### Pre-processed text (lowercase, punctuation/stop word removal):

*concerned lynch best lost highway dark violent surreal beautiful  
hallucinatory masterpiece 10 10 stars*

Decide what/how do you want to represent: Obtain a vocabulary

Assume a corpus  $D$  of  $m$  pre-processed texts (e.g. a set of movie reviews or tweets).

The vocabulary  $\mathcal{V}$  is a set containing all the  $k$  unique words  $w_i$  in  $D$ :

$$\mathcal{V} = \{w_1, \dots, w_k\}$$

and often is extended to include n-grams (contiguous sequences of  $n$  words).

## Words: Discrete vectors

Text:

*love pineapple apricot apple chocolate apple pie*

- $\mathcal{V} = \{ \text{apple}, \text{apricot}, \text{chocolate}, \text{love}, \text{pie}, \text{pineapple} \}$
- Vocabulary size:  $|\mathcal{V}| = 6$

$$\text{apricot} = \mathbf{x}_2$$

$$\text{pineapple} = \mathbf{x}_3$$

## Words: Discrete vectors

Text:

*love pineapple apricot apple chocolate apple pie*

- $\mathcal{V} = \{ \text{apple}, \text{apricot}, \text{chocolate}, \text{love}, \text{pie}, \text{pineapple} \}$
- Vocabulary size:  $|\mathcal{V}| = 6$

$$\text{apricot} = \mathbf{x}_2 = [0, 1, 0, 0, 0, 0]$$

$$\text{pineapple} = \mathbf{x}_3 = [0, 0, 0, 0, 0, 1]$$

## Words: Discrete vectors

Text:

*love pineapple apricot apple chocolate apple pie*

- $\mathcal{V} = \{ \text{apple}, \text{apricot}, \text{chocolate}, \text{love}, \text{pie}, \text{pineapple} \}$
- Vocabulary size:  $|\mathcal{V}| = 6$

$$\text{apricot} = \mathbf{x}_2 = [0, 1, 0, 0, 0, 0]$$

$$\text{pineapple} = \mathbf{x}_3 = [0, 0, 0, 0, 0, 1]$$

- Also known as **one-hot encoding**. What's the problem?

## Problems with discrete vectors

$$\text{apricot} = \mathbf{x}_2 = [0, 1, 0, 0, 0, 0]$$

$$\text{pineapple} = \mathbf{x}_3 = [0, 0, 0, 0, 0, 1]$$

$$\begin{aligned}\text{dot}(\mathbf{x}_2, \mathbf{x}_3) &= 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 1 \\ &= 0\end{aligned}$$

$$\text{cosine}(\mathbf{x}_2, \mathbf{x}_3) = \frac{\mathbf{x}_2 \cdot \mathbf{x}_3}{|\mathbf{x}_2| |\mathbf{x}_3|} = \frac{0}{1 \cdot 1} = 0$$

- Every word is equally different from every other word. But *apricot* and *pineapple* are related!
- Would **contextual information** be useful?

A quick test: What is **tesguino**?

# A quick test: What is **tesguino**?

Some sentences mentioning it:

- A bottle of *tesguino* is on the table.
- Everybody likes an ice cold *tesguino*.
- *Tesguino* makes you drunk.
- We make *tesguino* out of corn.

# A quick test: What is **tesguino**?

Some sentences mentioning it:

- A bottle of *tesguino* is on the table.
- Everybody likes an ice cold *tesguino*.
- *Tesguino* makes you drunk.
- We make *tesguino* out of corn.



*Tesguino is a beer made from corn.*

# Distributional Hypothesis

Firth (1957)<sup>1</sup>

You shall know a word by the company it keeps!

*Words appearing in similar contexts are likely to have similar meanings.*

---

<sup>1</sup> John R Firth (1957). "A synopsis of linguistic theory, 1930-1955". In: *Studies in linguistic analysis*.

## Words: Word-Word Matrix (term-context)

- A matrix  $X$ ,  $n \times m$  where  $n = |\mathcal{V}|$  (target words) and  $m = |\mathcal{V}_c|$  (context words)

## Words: Word-Word Matrix (term-context)

- A matrix  $X$ ,  $n \times m$  where  $n = |\mathcal{V}|$  (target words) and  $m = |\mathcal{V}_c|$  (context words)
- For each word  $x_i$  in  $\mathcal{V}$ , count how many times it co-occurs with context words  $x_j$

## Words: Word-Word Matrix (term-context)

- A matrix  $X$ ,  $n \times m$  where  $n = |\mathcal{V}|$  (target words) and  $m = |\mathcal{V}_c|$  (context words)
- For each word  $x_i$  in  $\mathcal{V}$ , count how many times it co-occurs with context words  $x_j$
- Use a context window of  $\pm k$  words (to the left/right of  $x_i$ )

## Words: Word-Word Matrix (term-context)

- A matrix  $X$ ,  $n \times m$  where  $n = |\mathcal{V}|$  (target words) and  $m = |\mathcal{V}_c|$  (context words)
- For each word  $x_i$  in  $\mathcal{V}$ , count how many times it co-occurs with context words  $x_j$
- Use a context window of  $\pm k$  words (to the left/right of  $x_i$ )
- Compute frequencies over a big corpus of documents (e.g. the entire Wikipedia)!

## Words: Word-Word Matrix (term-context)

- A matrix  $X$ ,  $n \times m$  where  $n = |\mathcal{V}|$  (target words) and  $m = |\mathcal{V}_c|$  (context words)
- For each word  $x_i$  in  $\mathcal{V}$ , count how many times it co-occurs with context words  $x_j$
- Use a context window of  $\pm k$  words (to the left/right of  $x_i$ )
- Compute frequencies over a big corpus of documents (e.g. the entire Wikipedia)!
- Usually target and context word vocabularies are the same resulting into a square matrix.

# Word-Word Matrix

sugar, a sliced lemon, a tablespoonful of their enjoyment. Cautiously she sampled her first well suited to programming on the digital for the purpose of gathering data and

**apricot**  
**pineapple**  
**computer.**  
**information**

preserve or jam, a pinch each of, and another fruit whose taste she likened In finding the optimal R-stage policy from necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

# Word-Word Matrix

sugar, a sliced lemon, a tablespoonful of their enjoyment. Cautiously she sampled her first well suited to programming on the digital for the purpose of gathering data and

apricot  
pineapple  
computer.  
information

preserve or jam, a pinch each of, and another fruit whose taste she likened In finding the optimal R-stage policy from necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

Now **apricot** and **pineapple** vectors look more **similar**!

- $\text{cosine}(\text{apricot}, \text{pineapple}) = 1$
- $\text{cosine}(\text{apricot}, \text{digital}) = 0$

# Context types

- We can refine contexts using linguistic information:
  - their part-of-speech tags (**bank\_V** vs. **bank\_N**)
  - syntactic dependencies (**eat\_dobj** vs. **eat\_subj**)
  - We will see how to extract this info soon!

## Documents: Document-Word Matrix (Bag-of-Words)

- A matrix  $X$ ,  $|D| \times |\mathcal{V}|$  where rows are documents in corpus  $D$ , and columns are vocabulary words in  $\mathcal{V}$ .
- For each document, count how many times words  $w \in \mathcal{V}$  appear in it.

## Documents: Document-Word Matrix (Bag-of-Words)

- A matrix  $X$ ,  $|D| \times |\mathcal{V}|$  where rows are documents in corpus  $D$ , and columns are vocabulary words in  $\mathcal{V}$ .
- For each document, count how many times words  $w \in \mathcal{V}$  appear in it.

	bad	good	great	terrible
Doc 1	14	1	0	5
Doc 2	2	5	3	0
Doc 3	0	2	5	0

- $X$  can also be obtained by adding all the one-hot vectors of the words in the documents and then transpose!

## Problems with counts

- Frequent words (articles, pronouns, etc.) dominate contexts without being informative.

## Problems with counts

- Frequent words (articles, pronouns, etc.) dominate contexts without being informative.
- Let's add the word **the** to the contexts. It often appears with most nouns:

vocabulary = [aadvark, computer, data, pinch, result, sugar, **the**]

apricot =  $\mathbf{x}_2 = [0, 0, 0, 1, 0, 1, 30]$

digital =  $\mathbf{x}_3 = [0, 2, 1, 0, 1, 0, 45]$

$$\text{cosine}(\mathbf{x}_2, \mathbf{x}_3) = \frac{30 \cdot 45}{\sqrt{902} \cdot \sqrt{2031}} = 0.997$$

## Problems with counts

- Frequent words (articles, pronouns, etc.) dominate contexts without being informative.
- Let's add the word **the** to the contexts. It often appears with most nouns:

vocabulary = [aadvark, computer, data, pinch, result, sugar, **the**]

apricot =  $\mathbf{x}_2 = [0, 0, 0, 1, 0, 1, 30]$

digital =  $\mathbf{x}_3 = [0, 2, 1, 0, 1, 0, 45]$

$$\text{cosine}(\mathbf{x}_2, \mathbf{x}_3) = \frac{30 \cdot 45}{\sqrt{902} \cdot \sqrt{2031}} = 0.997$$

- This also holds for the document-word matrix! Solution:  
**Weight the vectors!**

## Weighting the Word-Word Matrix: Distance discount

Weight contexts according to the distance from the word: the further away, the lower the weight!

- For a window size  $\pm k$ , multiply the context word at each position as  $\frac{k - \text{distance}}{k}$ , e.g. for  $k = 3$ :

$$\left[ \frac{1}{3}, \frac{2}{3}, \frac{3}{3}, \textit{word}, \frac{3}{3}, \frac{2}{3}, \frac{1}{3} \right]$$

## Weighting the Word-Word Matrix: Pointwise Mutual Information

**Pointwise Mutual Information (PMI):** how often two words  $w_i$  and  $w_j$  occur together relative to occur independently:

$$\text{PMI}(w_i, w_j) = \log_2 \frac{P(w_i, w_j)}{P(w_i)P(w_j)} = \frac{\#(w_i, w_j)|D|}{\#(w_i) \cdot \#(w_j)}$$
$$P(w_i, w_j) = \frac{\#(w_i, w_j)}{|D|}, P(w_i) = \frac{\#(w_i)}{|D|}$$

where  $\#(\cdot)$  denotes count and  $|D|$  number of observed word-context word pairs in the corpus.

# Weighting the Word-Word Matrix: Pointwise Mutual Information

**Pointwise Mutual Information (PMI):** how often two words  $w_i$  and  $w_j$  occur together relative to occur independently:

$$PMI(w_i, w_j) = \log_2 \frac{P(w_i, w_j)}{P(w_i)P(w_j)} = \frac{\#(w_i, w_j)|D|}{\#(w_i) \cdot \#(w_j)}$$
$$P(w_i, w_j) = \frac{\#(w_i, w_j)}{|D|}, P(w_i) = \frac{\#(w_i)}{|D|}$$

where  $\#(\cdot)$  denotes count and  $|D|$  number of observed word-context word pairs in the corpus.

- Positive values quantify relatedness. Use PMI instead of counts.
- Negative values? Usually ignored (positive PMI):

$$PPMI(w_i, w_j) = \max(PMI(w_i, w_j), 0)$$

## Weighting the Document-Word Matrix: TF.IDF

- Penalise words appearing in many documents.
- Multiply word frequencies with their inverted document frequencies:

$$idf_w = \log_{10} \frac{N}{df_w}$$

where  $N$  is the number of documents in the corpus,  $df_w$  is document frequency of word  $w$

## Weighting the Document-Word Matrix: TF.IDF

- Penalise words appearing in many documents.
- Multiply word frequencies with their inverted document frequencies:

$$idf_w = \log_{10} \frac{N}{df_w}$$

where  $N$  is the number of documents in the corpus,  $df_w$  is document frequency of word  $w$

- To obtain:

$$x_{id} = tf_{id} \log_{10} \frac{N}{df_{id}}$$

- We can also squash the raw frequency ( $tf$ ), by using its  $\log_{10}$ .

# Problems with Dimensionality

- Count-based matrices (for words and documents) often work well, but:
  - high dimensional: vocabulary size could be millions!
  - very sparse: words co-occur only with a small number of words; documents contain only a very small subset of the vocabulary

# Problems with Dimensionality

- Count-based matrices (for words and documents) often work well, but:
  - high dimensional: vocabulary size could be millions!
  - very sparse: words co-occur only with a small number of words; documents contain only a very small subset of the vocabulary
- Solution: **Dimensionality Reduction to the rescue!**

# Truncated Singular Value Decomposition

- A method for finding the most important dimensions of a data set, those dimensions along which the data varies the most by decomposing the matrix into latent factors.
- Truncated Singular Value Decomposition (truncated-SVD):

$$X^{n \times m} \approx U^{n \times k} S^{k \times k} V^{k \times m}$$

- Approximation is good: exploits redundancy to remove noise by learning a low-dimensional latent space.
- For a detailed description see this [tutorial](#).

# Singular Value Decomposition on Word-Word matrix

1) SVD

$$\text{word-word PPMI matrix } \mathbf{X} = \mathbf{W} \mathbf{S} \mathbf{C}$$

$\mathbf{X}$        $\mathbf{W}$        $\mathbf{S}$        $\mathbf{C}$

$w \times c$        $w \times m$        $m \times m$        $m \times c$

2) Truncation:

$$\approx \mathbf{W} \mathbf{S} \mathbf{C}$$

$\mathbf{W}$        $\mathbf{S}$        $\mathbf{C}$

$w \times m$        $m \times m$        $m \times c$

~~$m \times m$~~        ~~$m \times m$~~        ~~$m \times c$~~

~~$k$~~        ~~$k$~~        ~~$k$~~

3) Embeddings:

embedding for word i:

$$\mathbf{W}_k$$

$1, \dots, k$

$i$

$2$

$.$

$w$

$w \times k$

$\mathbf{S}$        $\mathbf{C}$

# Singular Value Decomposition on Document-Word matrix

- Also called Latent Semantic Analysis<sup>2</sup> (LSA)
- $U^{n \times k}$  represents document embeddings
- $V^{k \times m}$  represents word embeddings
- You can obtain an embedding  $\mathbf{u}_{\text{new}}$  for a new document  $\mathbf{x}_{\text{new}}$  by projecting its count vector to the latent space:

$$\mathbf{u}_{\text{new}} = \mathbf{x}_{\text{new}} V_k^\top$$

---

<sup>2</sup>Susan T Dumais et al. (1988). "Using latent semantic analysis to improve access to textual information". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 281–285.

# Evaluation: Word Vectors

- Intrinsic:
  - similarity: order word pairs according to their semantic similarity
  - in-context similarity: substitute a word in a sentence without changing its meaning.
  - analogy: Athens is to Greece what Rome is to ...?

# Evaluation: Word Vectors

- Intrinsic:
  - similarity: order word pairs according to their semantic similarity
  - in-context similarity: substitute a word in a sentence without changing its meaning.
  - analogy: Athens is to Greece what Rome is to ...?
- Extrinsic: use them to improve performance in a task, i.e. instead of bag of words → **bag of word vectors (embeddings)**

## Best word vectors?

- high-dimensional count-based?
- low-dimensional with SVD? (In Lecture 6, we will see how we can obtain low-dimensional vectors with Neural Networks)
- Levy et al.<sup>3</sup> (2015) showed that choice of context window size, rare word removal, etc. matter more.
- Choice of texts to obtain the counts matters. More text is better, and low-dimensional methods scale better.

---

<sup>3</sup>Omer Levy, Yoav Goldberg, and Ido Dagan (2015). "Improving Distributional Similarity with Lessons Learned from Word Embeddings". In: *Transactions of the Association for Computational Linguistics*, pp. 211–225.

## Limitations: Word Vectors

- **Polysemy:** All occurrences of a word (and all its senses) are represented by one vector.
  - Given a task, it is often useful to adapt the word vectors to represent the appropriate sense
- **Antonyms** appear in similar contexts, hard to distinguish them from synonyms
- **Compositionality:** what is the meaning of a sequence of words?
  - while we might be able to obtain context vectors for short phrases, this doesn't scale to whole sentences, paragraphs, etc.
  - Solution: combine word vectors, i.e. add/multiply
  - Soon we will see methods to learn embeddings for word sequences from word embeddings, the recurrent neural networks!

# Evaluation: Document Vectors

- Intrinsic:
  - document similarity
  - information retrieval

# Evaluation: Document Vectors

- Intrinsic:
  - document similarity
  - information retrieval
- Extrinsic:
  - text classification, plagiarism detection etc.

## Limitations: Document Vectors

- Word order is ignored, but language is sequential!

## Upcoming next...

- Document vectors + machine learning → text classification!

# Reading Material

- 6-1 to 6-7, 6-9 to 6-12 from [Chapter 6] Jurafsky & Martin
- Vector space models of semantics [paper]
- References in slides

# Dependency Parsing

## COM6513 Natural Language Processing

Nikos Aletras

[n.aletras@sheffield.ac.uk](mailto:n.aletras@sheffield.ac.uk)

@nikaletras

Computer Science Department

Week 5  
Spring 2021



In previous lectures...

- **Text Classification:** Given an instance  $x$  (e.g. document), predict a label  $y \in \mathcal{Y}$
- **Tasks:** sentiment analysis, topic classification, etc.
- **Algorithm:** Logistic Regression

## In previous lectures...

- **Sequence labelling:** Given a sequence of words  $x = [x_1, \dots, x_N]$ , predict a sequence of labels  $y \in \mathcal{Y}^N$
- **Tasks:** part of speech tagging, named entity recognition, etc.
- **Algorithms:** Hidden Markov Models, Conditional Random Fields

## In this lecture...

- Model richer linguistic representations: **graphs**
- **Dependency parses:** Graphs representing syntactic relations between words in a sentence

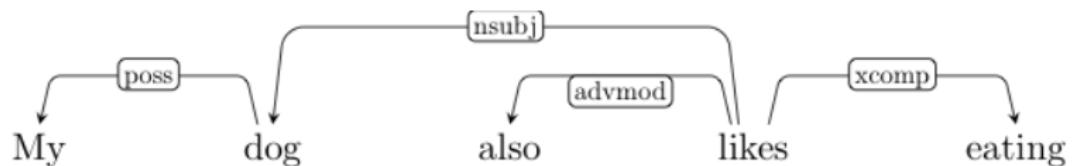
## In this lecture...

- Model richer linguistic representations: **graphs**
- **Dependency parses:** Graphs representing syntactic relations between words in a sentence
- Two approaches:
  - Graph-based Dependency Parsing
  - Transition-based Dependency Parsing

# Dependency Parsing: Applications

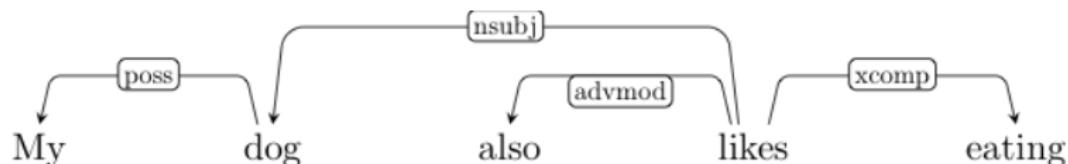
- **Relation extraction**, e.g. identify entity pairs (AM, Arctic Monkeys), (Abbey Road, Beatles), (Different Class, Pulp) with the relation `music_album_by`
- **Question answering**
- **Sentiment analysis**

# What is a Dependency Parse?



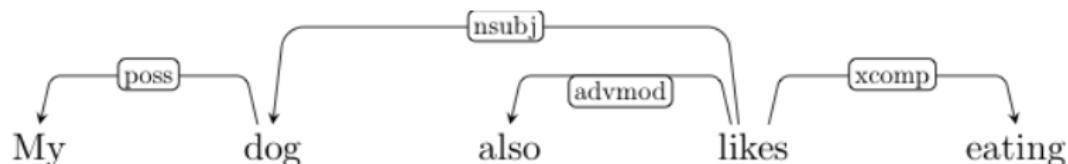
- **Dependency parse (or tree):** Graph representing syntactic relations between words in a sentence

# What is a Dependency Parse?



- **Dependency parse (or tree):** Graph representing syntactic relations between words in a sentence
- **Nodes (or vertices):** Words in a sentence
- **Edges (or arcs):** Syntactic relations between words, e.g. dog is the subject (nsbj) of likes ([list of standard dependency relations](#))

# What is a Dependency Parse?



- **Dependency parse (or tree):** Graph representing syntactic relations between words in a sentence
- **Nodes (or vertices):** Words in a sentence
- **Edges (or arcs):** Syntactic relations between words, e.g. dog is the subject (nsubj) of likes ([list of standard dependency relations](#))
- **Dependency Parsing:** Automatically identify the syntactic relations between words in a sentence

## Graph constraints

- **Connected:** every word can be reached from any other word ignoring edge directionality

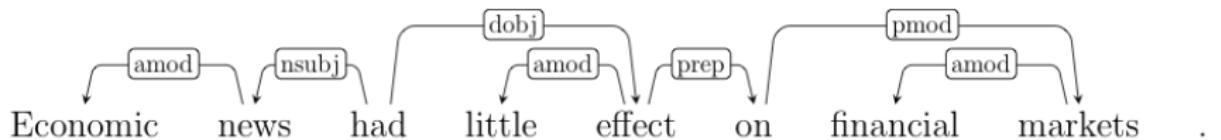
# Graph constraints

- **Connected:** every word can be reached from any other word ignoring edge directionality
- **Acyclic:** can't re-visit the same word on a directed path

# Graph constraints

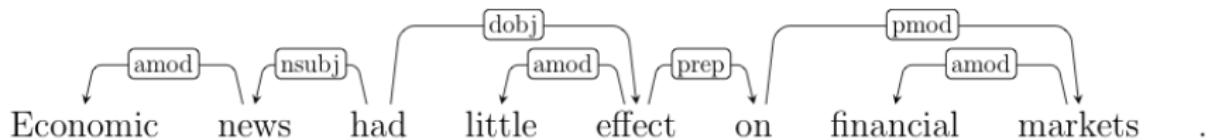
- **Connected:** every word can be reached from any other word ignoring edge directionality
- **Acyclic:** can't re-visit the same word on a directed path
- **Single-Head:** every word can have only one head

# Well-formed Dependency Parse?



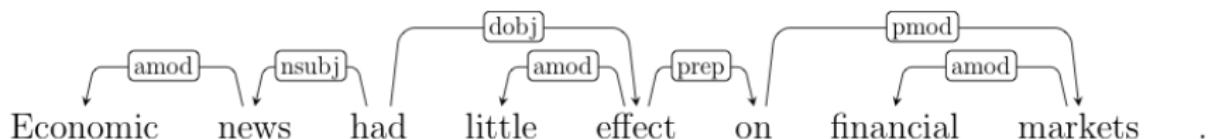
- Connected?

# Well-formed Dependency Parse?



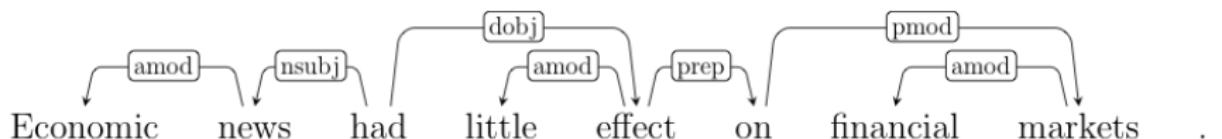
- Connected? NO

# Well-formed Dependency Parse?



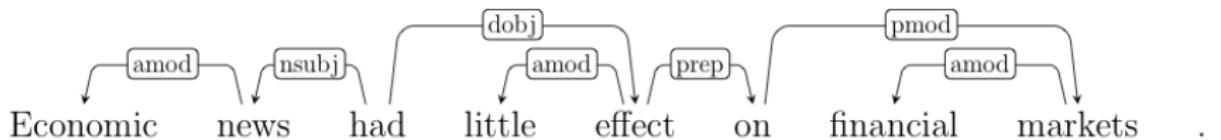
- Connected? NO
- Acyclic?

# Well-formed Dependency Parse?



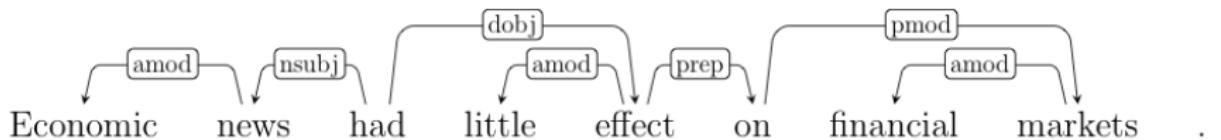
- Connected? NO
- Acyclic? YES

# Well-formed Dependency Parse?



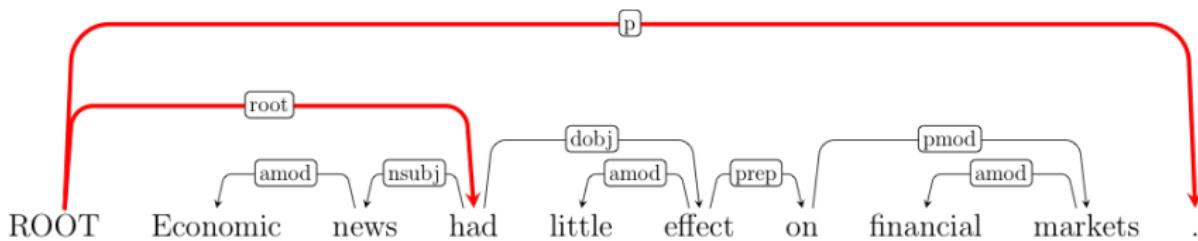
- Connected? NO
- Acyclic? YES
- Single-headed?

# Well-formed Dependency Parse?



- Connected? NO
- Acyclic? YES
- Single-headed? YES
- Solution?

# Well-formed Dependency Parse



Add a **special root node** with edges to any nodes without heads  
(main verb and punctuation).

# Dependency Parsing: Problem setup

Training data is pairs of word sequences (sentences) and dependency trees:

$$D_{train} = \{(\mathbf{x}^1, G_x^1), \dots, (\mathbf{x}^M, G_x^M)\}$$

$$\mathbf{x}^m = [x_1, \dots, x_N]$$

$$\text{graph } G_x = (V_x, A_x)$$

$$\text{vertices } V_x = \{0, 1, \dots, N\}$$

$$\text{edges } A_x = \{(i, j, k) | i, j \in V, k \in L(\text{labels})\}$$

# Dependency Parsing: Problem setup

Training data is pairs of word sequences (sentences) and dependency trees:

$$D_{train} = \{(\mathbf{x}^1, G_x^1), \dots, (\mathbf{x}^M, G_x^M)\}$$

$$\mathbf{x}^m = [x_1, \dots, x_N]$$

$$\text{graph } G_x = (V_x, A_x)$$

$$\text{vertices } V_x = \{0, 1, \dots, N\}$$

$$\text{edges } A_x = \{(i, j, k) | i, j \in V, k \in L(\text{labels})\}$$

We want to learn a model to predict the best graph:

$$\hat{G}_x = \arg \max_{G_x \in \mathcal{G}_x} \text{score}(G_x, \mathbf{x})$$

# Learning a dependency parser

We want to learn a model to predict the best graph:

$$\hat{G}_x = \arg \max_{G_x \in \mathcal{G}_x} \text{score}(G_x, x)$$

where the  $G_x$  is a well-formed dependency tree.

- **Can we learn it using what we know so far?**

# Learning a dependency parser

We want to learn a model to predict the best graph:

$$\hat{G}_x = \arg \max_{G_x \in \mathcal{G}_x} score(G_x, x)$$

where the  $G_x$  is a well-formed dependency tree.

- **Can we learn it using what we know so far?** Enumeration over all possible graphs will be expensive.

# Learning a dependency parser

We want to learn a model to predict the best graph:

$$\hat{G}_x = \arg \max_{G_x \in \mathcal{G}_x} \text{score}(G_x, x)$$

where the  $G_x$  is a well-formed dependency tree.

- **Can we learn it using what we know so far?** Enumeration over all possible graphs will be expensive.
- **How about a classifier that predicts each edge?**

# Learning a dependency parser

We want to learn a model to predict the best graph:

$$\hat{G}_x = \arg \max_{G_x \in \mathcal{G}_x} \text{score}(G_x, x)$$

where the  $G_x$  is a well-formed dependency tree.

- **Can we learn it using what we know so far?** Enumeration over all possible graphs will be expensive.
- **How about a classifier that predicts each edge?** Maybe. But predicting an edge makes some edges invalid due to the acyclic and single-head constraints.

# Maximum Spanning Tree

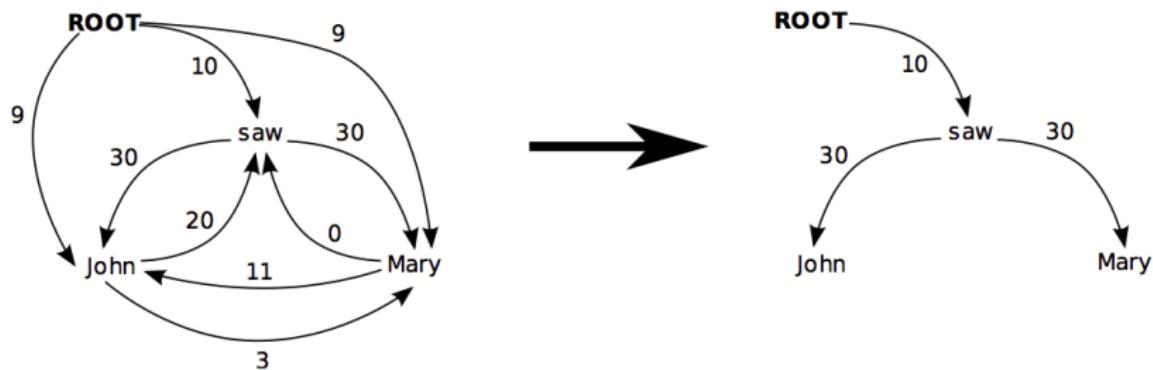
- **Spanning Tree:** In graph theory, a spanning tree  $T$  of an undirected graph  $G$  is a subgraph that includes all of the vertices of  $G$ , with the minimum possible number of edges.

# Maximum Spanning Tree

- **Spanning Tree:** In graph theory, a spanning tree  $T$  of an undirected graph  $G$  is a subgraph that includes all of the vertices of  $G$ , with the minimum possible number of edges.
- **Tree:** In computer science, a tree is a widely used data structure (Abstract Data Type) that simulates a hierarchical tree structure, with a root value and subtrees of children with a parent node, represented as a set of linked nodes.

# Maximum Spanning Tree

Score all edges, but keep only the max spanning tree using Chu-Liu-Edmonds algorithm, a modification to Kruskal's algorithm for extracting Maximum Spanning Trees.



Exact solution in  $O(N^2)$  time using Chu-Liu-Edmonds.

# Kruskal's algorithm

```
Input scored edges  $E$ 
      sort  $E$  by cost (opposite of score)
       $G = \{\}$ 
      while  $G$  not spanning do:
          pop the next edge  $e$ 
          if connecting different trees :
              add  $e$  to  $G$ 
      Return  $G$ 
```

# Graph-based Dependency Parsing

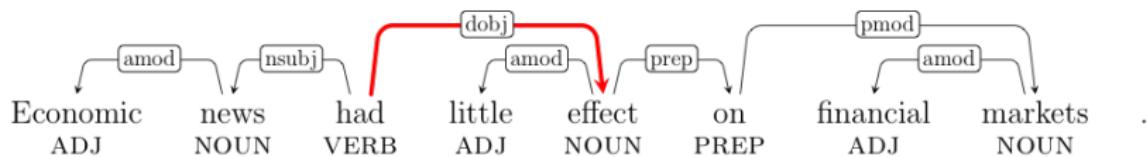
Decompose the graph score into arc scores:

$$\begin{aligned}\hat{G}_x &= \arg \max_{G_x \in \mathcal{G}_x} \text{score}(G_x, x) \\ &= \arg \max_{G_x \in \mathcal{G}_x} \mathbf{w} \cdot \Phi(G_x, x) \quad (\text{linear model}) \\ &= \arg \max_{G_x \in \mathcal{G}_x} \sum_{(i,j,l) \in A_x} \mathbf{w} \cdot \phi((i,j,l), x) \quad (\text{arc-factored})\end{aligned}$$

Can learn the weights with a Conditional Random Field!

# Feature Representation

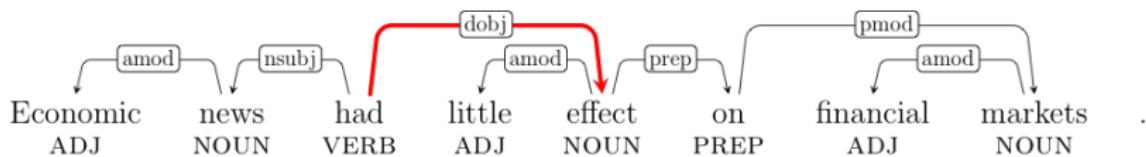
What should  $\phi((head, dependent, label), \mathbf{x})$  be?



- **unigram:** head=had, head=VERB

# Feature Representation

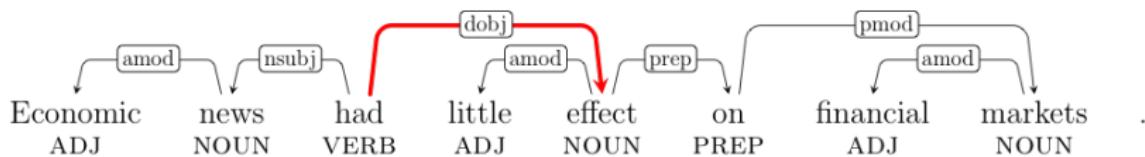
What should  $\phi((head, dependent, label), \mathbf{x})$  be?



- **unigram:** head=had, head=VERB
- **bigram:** head=had & dependent=effect

# Feature Representation

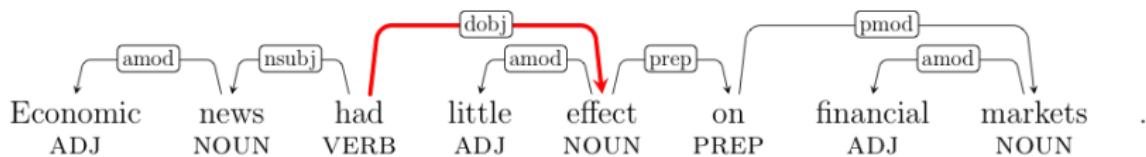
What should  $\phi((\text{head}, \text{dependent}, \text{label}), \mathbf{x})$  be?



- **unigram:** head=had, head=VERB
- **bigram:** head=had & dependent=effect
- head=VERB & dependent=NOUN & between=ADJ

# Feature Representation

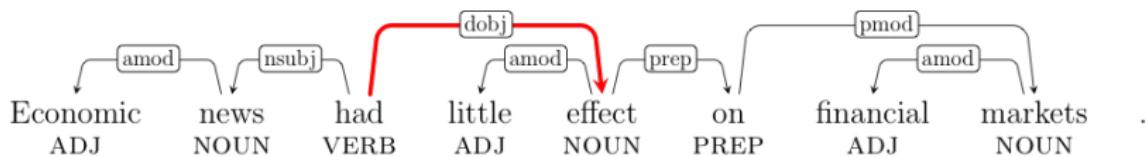
What should  $\phi((head, dependent, label), \mathbf{x})$  be?



- **unigram:** head=had, head=VERB
- **bigram:** head=had & dependent=effect
- head=VERB & dependent=NOUN & between=ADJ
- head=had & label=dobj & other-label=nsubj

# Feature Representation

What should  $\phi((head, dependent, label), \mathbf{x})$  be?



- **unigram:** head=had, head=VERB
- **bigram:** head=had & dependent=effect
- head=VERB & dependent=NOUN & between=ADJ
- head=had & label=dobj & other-label=nsubj  
**NO!!! Breaks the arc-factored scoring**

## More Global models

- Even though inference and learning are global, features are localised to arcs.
- **Can we have more global features?**

## More Global models

- Even though inference and learning are global, features are localised to arcs.
- **Can we have more global features?** Yes we can! Consider subgraphs spanning a few edges. But inference becomes harder, requiring more complex dynamic programs and clever approximations.
- **Is it worth it?** Syntactic parsing has many applications, thus better compromises between speed and accuracy are always welcome!

# Transition-based Dependency Parsing

- Graph-based dependency parsing restricts the features to perform joint inference efficiently.

# Transition-based Dependency Parsing

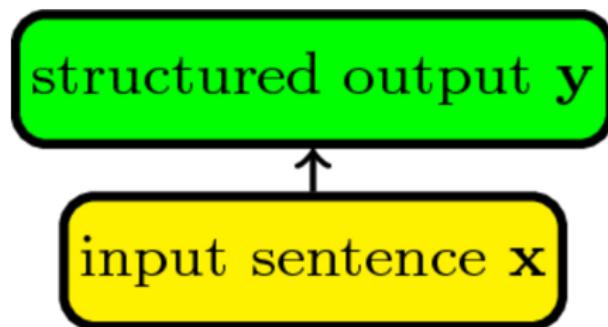
- Graph-based dependency parsing restricts the features to perform joint inference efficiently.
- **Transition-based dependency parsing** trades joint inference for feature flexibility.

# Transition-based Dependency Parsing

- Graph-based dependency parsing restricts the features to perform joint inference efficiently.
- **Transition-based dependency parsing** trades joint inference for feature flexibility.
- No more argmax over graphs, just use a classifier with any features we want!

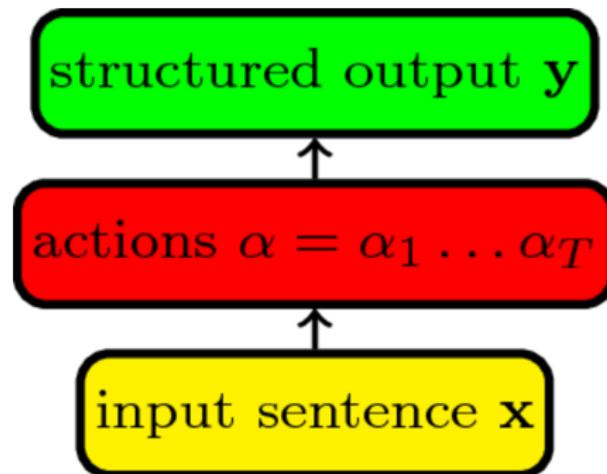
# Joint vs incremental prediction

**Joint:** score (and enumerate) complete outputs (graphs)



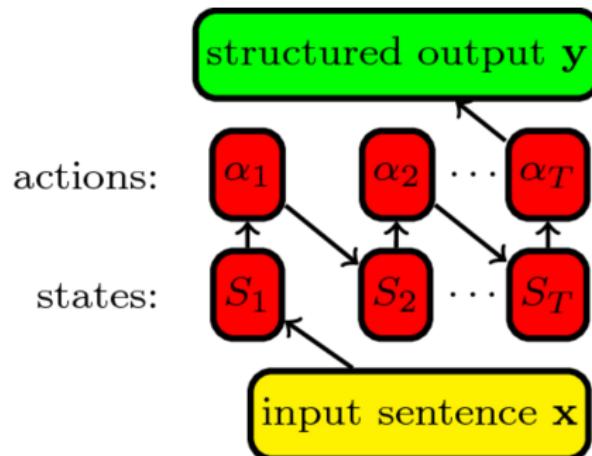
## Joint vs incremental prediction

**Incremental:** predict a sequence of actions (transitions)  
constructing the output



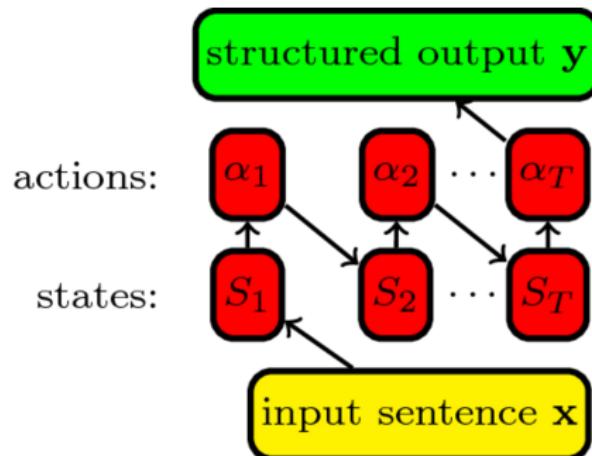
# Transition system

The **actions**  $\mathcal{A}$  the classifier  $f$  can predict and their effect on the **state** which tracks the prediction:  $S_{t+1} = S_1(\alpha_1 \dots \alpha_t)$



# Transition system

The **actions**  $\mathcal{A}$  the classifier  $f$  can predict and their effect on the **state** which tracks the prediction:  $S_{t+1} = S_1(\alpha_1 \dots \alpha_t)$



What should the actions (transitions) be for dependency parsing?

# Transition system setup

- **Input:** Vertices  $V_x = \{0, 1, \dots, N\}$  (words sentence  $x$ )
- **State**  $S = (Stack, B, A)$ :
  - Arcs  $A$  (dependencies predicted so far)
  - Buffer  $Buf$  (words left to process)
  - Stack  $Stack$  (last-in, first out memory)
- Initial state:  $S_0 = ([], [0, 1, \dots, N], \{\})$
- Final state:  $S_{final} = (Stack, [], A)$

## Transition system

Shift ( $Stack, i | Buf, A$ )  $\rightarrow$  ( $Stack | i, Buf, A$ ): push next word from the buffer ( $i$ ) to stack

## Transition system

Shift ( $Stack, i | Buf, A$ )  $\rightarrow$  ( $Stack | i, Buf, A$ ): push next word from the buffer ( $i$ ) to stack

Reduce ( $Stack | i, Buf, A$ )  $\rightarrow$  ( $Stack, Buf, A$ ): pop word top of the stack ( $i$ ) if it has a head

## Transition system

Shift ( $Stack, i|Buf, A$ )  $\rightarrow$  ( $Stack|i, Buf, A$ ): push next word from the buffer ( $i$ ) to stack

Reduce ( $Stack|i, Buf, A$ )  $\rightarrow$  ( $Stack, Buf, A$ ): pop word top of the stack ( $i$ ) if it has a head

Right-Arc( $label$ ) ( $Stack|i,j|Buf, A$ )  $\rightarrow$  ( $Stack|i|j, Buf, A \cup \{(i,j,l)\}$ ): create edge  $(i,j,label)$  between top of the stack ( $i$ ) and next in buffer ( $j$ ), push  $j$

## Transition system

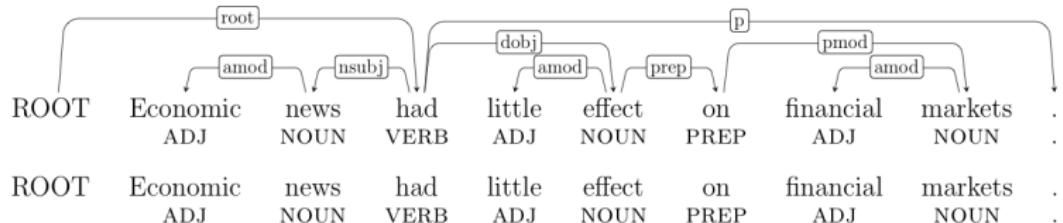
Shift ( $Stack, i|Buf, A$ )  $\rightarrow$  ( $Stack|i, Buf, A$ ): push next word from the buffer ( $i$ ) to stack

Reduce ( $Stack|i, Buf, A$ )  $\rightarrow$  ( $Stack, Buf, A$ ): pop word top of the stack ( $i$ ) if it has a head

Right-Arc( $label$ ) ( $Stack|i, j|Buf, A$ )  $\rightarrow$  ( $Stack|i|j, Buf, A \cup \{(i, j, l)\}$ ): create edge  $(i, j, label)$  between top of the stack ( $i$ ) and next in buffer ( $j$ ), push  $j$

Left-Arc( $label$ ) ( $Stack|i, j|Buf, A$ )  $\rightarrow$  ( $Stack, j|Buf, A \cup \{(j, i, l)\}$ ): create edge  $(j, i, label)$  and pop  $i$ , if  $i$  has no head

# Example

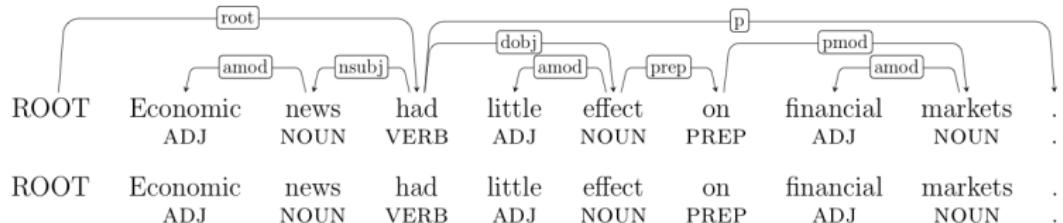


Stack = []

Buffer = [ROOT, Economic, news, had, little, effect, on, financial, markets, .]

Action?

# Example

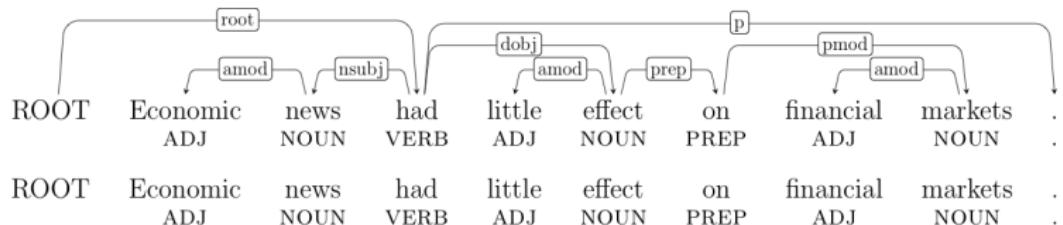


Stack = []

Buffer = [ROOT, Economic, news, had, little, effect, on, financial, markets, .]

Action? Shift

# Example

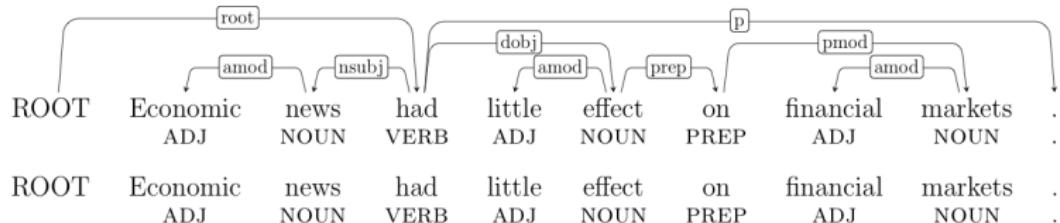


Stack = [ROOT]

Buffer = [Economic, news, had, little, effect, on, financial, markets, .]

Action?

# Example

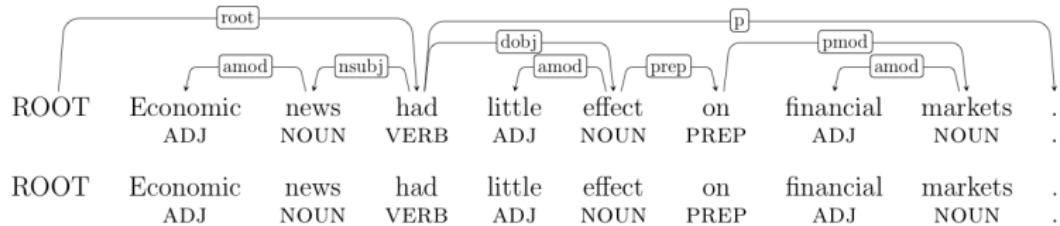


Stack = [ROOT]

Buffer = [Economic, news, had, little, effect, on, financial, markets, .]

Action? Shift

# Example

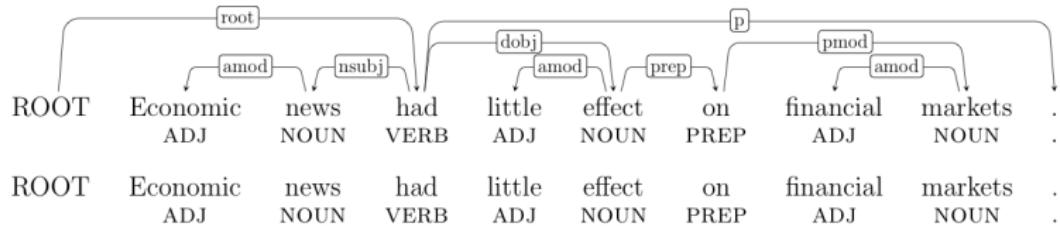


Stack = [ROOT, Economic]

Buffer = [news, had, little, effect, on, financial, markets, .]

Action?

# Example

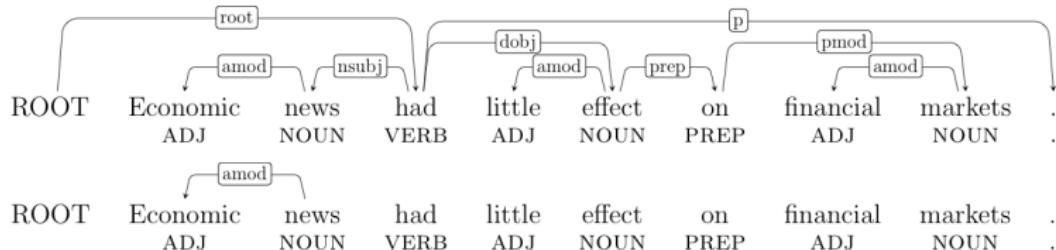


Stack = [ROOT, Economic]

Buffer = [news, had, little, effect, on, financial, markets, .]

Action? Left-Arc(*amod*)

# Example

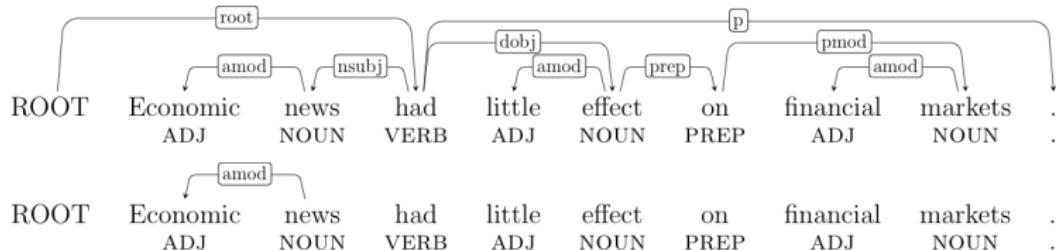


Stack = [ROOT]

Buffer = [news, had, little, effect, on, financial, markets, .]

Action?

# Example

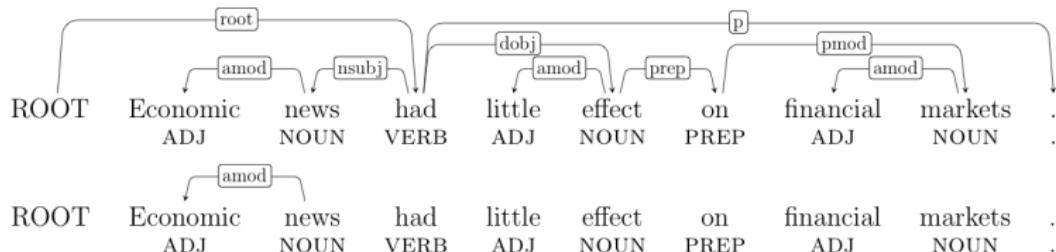


Stack = [ROOT]

Buffer = [news, had, little, effect, on, financial, markets, .]

Action? Shift

# Example

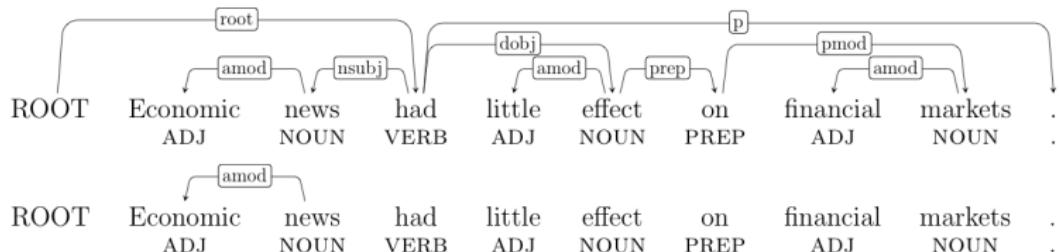


Stack = [ROOT, news]

Buffer = [had, little, effect, on, financial, markets, .]

Action?

# Example

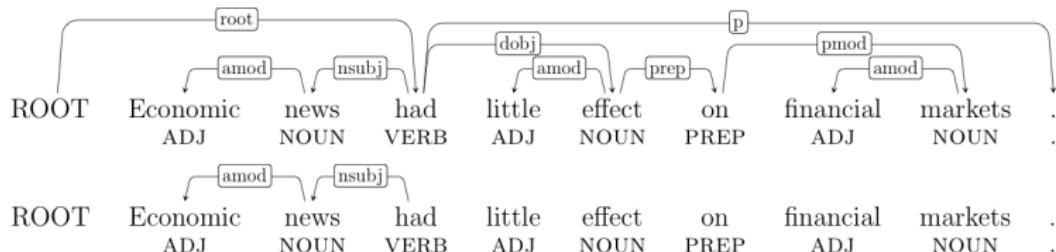


Stack = [ROOT, news]

Buffer = [had, little, effect, on, financial, markets, .]

Action? Left-Arc(*nsubj*)

# Example

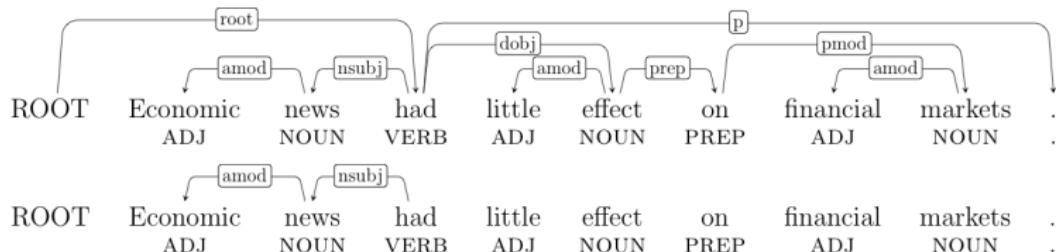


Stack = [ROOT]

Buffer = [had, little, effect, on, financial, markets, .]

Action?

# Example

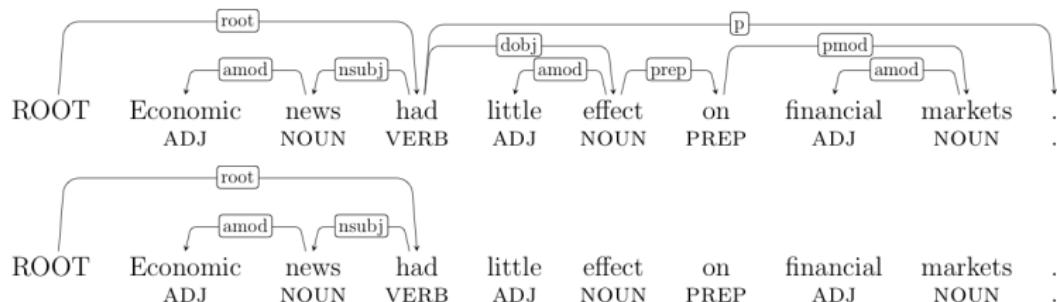


Stack = [ROOT]

Buffer = [had, little, effect, on, financial, markets, .]

Action? Right-Arc(*root*)

# Example

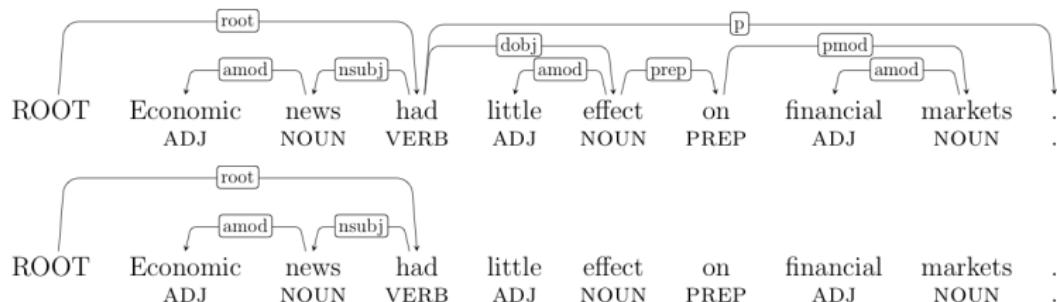


Stack = [ROOT, had]

Buffer = [little, effect, on, financial, markets, .]

Action?

# Example

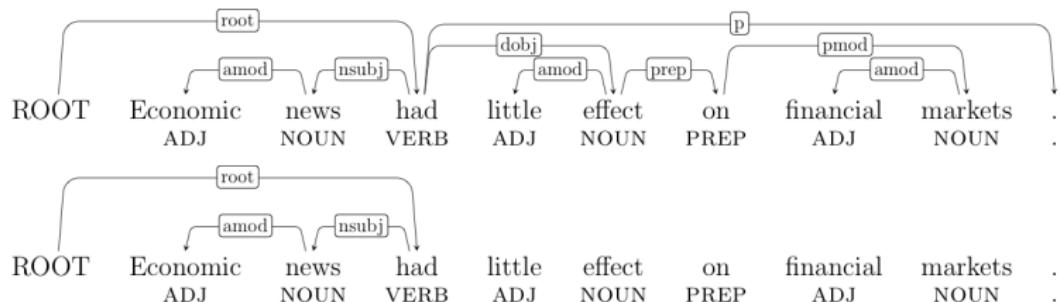


Stack = [ROOT, had]

Buffer = [little, effect, on, financial, markets, .]

Action? Shift

# Example

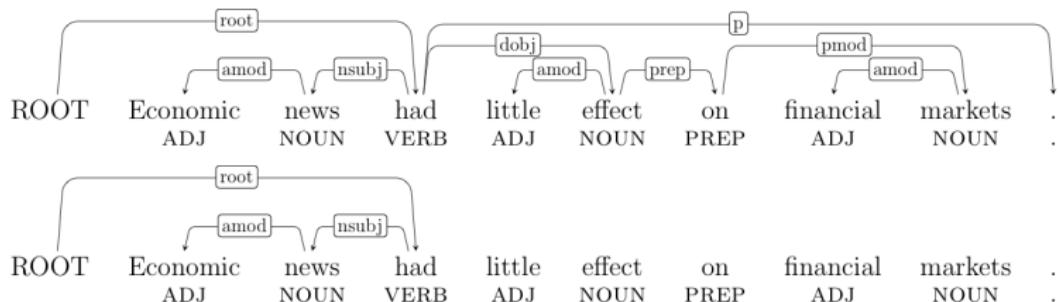


Stack = [ROOT, had, little]

Buffer = [effect, on, financial, markets, .]

Action?

# Example

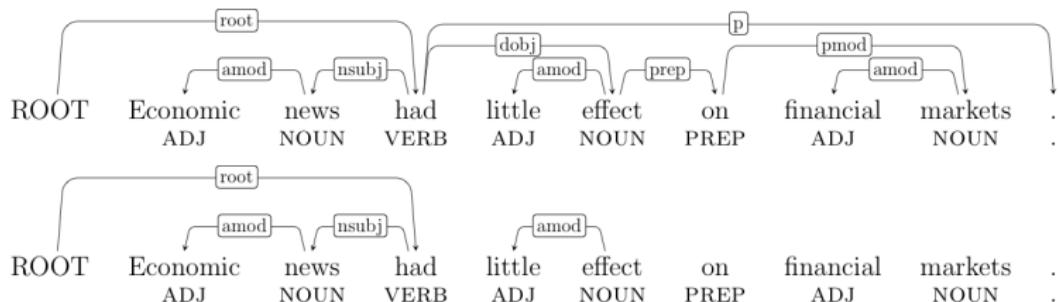


Stack = [ROOT, had, little]

Buffer = [effect, on, financial, markets, .]

Action?    Left-Arc(*amod*)

# Example

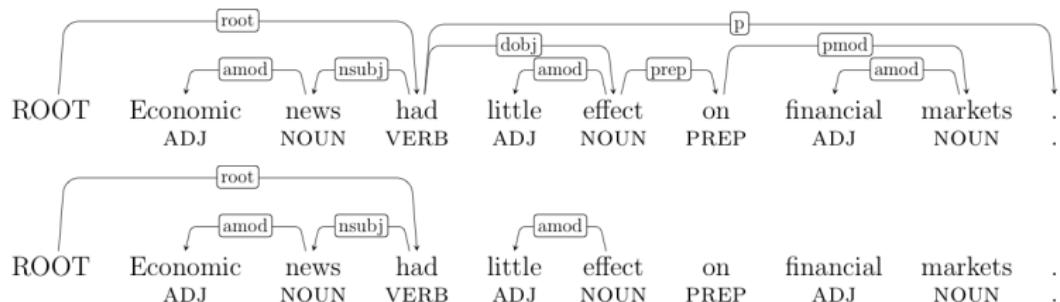


Stack = [ROOT, had]

Buffer = [effect, on, financial, markets, .]

Action?

# Example

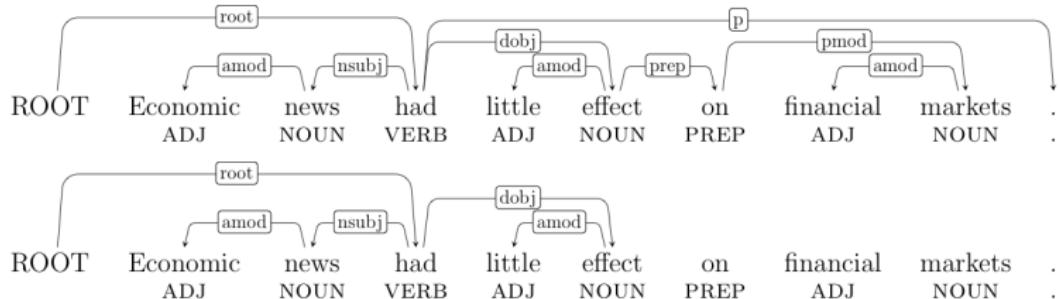


Stack = [ROOT, had]

Buffer = [effect, on, financial, markets, .]

Action? Right-Arc(*dobj*)

# Example

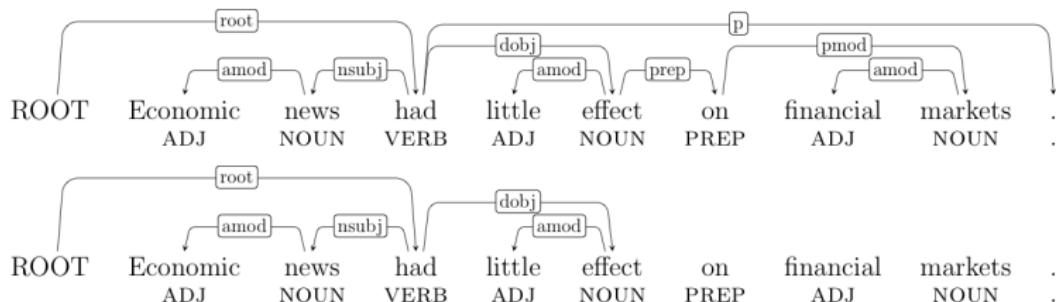


Stack = [ROOT, had, effect]

Buffer = [on, financial, markets, .]

Action?

# Example

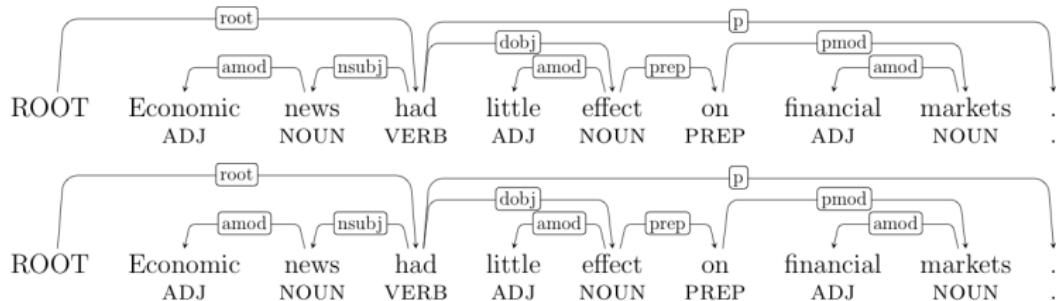


Stack = [ROOT, had, effect]

Buffer = [on, financial, markets, .]

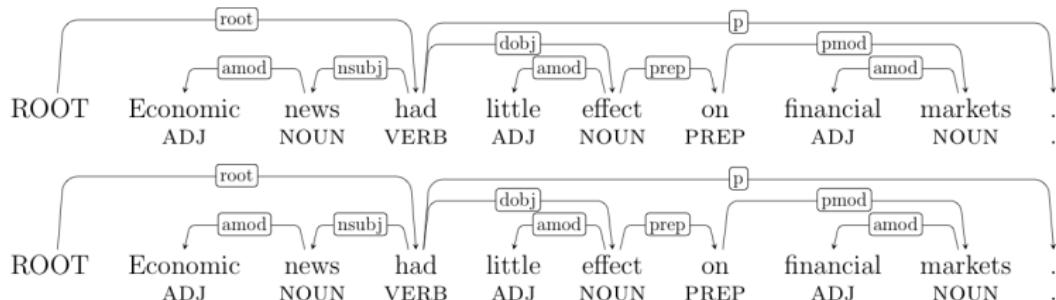
Action? let's fast-forward...

# Example



Empty buffer.

# Example



Stack = [ROOT, had, .]

Buffer = []

Empty buffer. DONE!

## Other transition systems?

- This was the arc-eager system. Others:

## Other transition systems?

- This was the arc-eager system. Others:
  - arc-standard (3 actions)

# Other transition systems?

- This was the arc-eager system. Others:
  - arc-standard (3 actions)
  - easy-first (not left-to-right), etc.
- All operate with actions combining:
  - moving words from the buffer to the stack and back (shift/un-shift)
  - popping words from the stack (reduce)
  - creating labeled arcs left and right

# Other transition systems?

- This was the arc-eager system. Others:
  - arc-standard (3 actions)
  - easy-first (not left-to-right), etc.
- All operate with actions combining:
  - moving words from the buffer to the stack and back (shift/un-shift)
  - popping words from the stack (reduce)
  - creating labeled arcs left and right
- Intuition: Define actions that are easy to learn

# Transition-based Dependency Parsing

**Input:** sentence  $\mathbf{x}$

state  $S_1 = \text{initialize}(\mathbf{x})$ ; timestep  $t = 1$

**while**  $S_t$  not final **do**

action  $\alpha_t = \arg \max_{\alpha \in \mathcal{A}} f(\alpha, S_t)$

$S_{t+1} = S_t(\alpha_t)$ ;  $t = t + 1$

What is  $f$ ?

# Transition-based Dependency Parsing

**Input:** sentence  $\mathbf{x}$

state  $S_1 = \text{initialize}(\mathbf{x})$ ; timestep  $t = 1$

**while**  $S_t$  not final **do**

action  $\alpha_t = \arg \max_{\alpha \in \mathcal{A}} f(\alpha, S_t)$

$S_{t+1} = S_t(\alpha_t)$ ;  $t = t + 1$

What is  $f$ ? A multiclass classifier

What do we need to learn it?

# Transition-based Dependency Parsing

**Input:** sentence  $\mathbf{x}$

$state S_1 = initialize(\mathbf{x}); timestep t = 1$

**while**  $S_t$  not final **do**

$action \alpha_t = \arg \max_{\alpha \in \mathcal{A}} f(\alpha, S_t)$

$S_{t+1} = S_t(\alpha_t); t = t + 1$

What is  $f$ ? A multiclass classifier

What do we need to learn it?

- learning algorithm (e.g. logistic regression)
- labelled training data
- feature representation

# What are the right actions?



We only have sentences labelled with graphs:

$$D_{train} = \{(\mathbf{x}^1, G_x^1) \dots (\mathbf{x}^M, G_x^M)\}$$

# What are the right actions?



We only have sentences labelled with graphs:

$$D_{train} = \{(\mathbf{x}^1, G_x^1) \dots (\mathbf{x}^M, G_x^M)\}$$

Ask an oracle to tell us the actions constructing the graph!

# What are the right actions?



We only have sentences labelled with graphs:

$$D_{train} = \{(\mathbf{x}^1, G_x^1) \dots (\mathbf{x}^M, G_x^M)\}$$

Ask an oracle to tell us the actions constructing the graph!

In our case, a set of **rules** comparing the current state  $S = (Stack, Buffer, ArcsPredicted)$  with  $G_x$  returning the correct action as label

# Learning from an oracle

Given a labelled sentence and a transition system, an oracle returns states labelled with the correct actions.

$$D_{train} = \{(\mathbf{x}^1, G_x^1) \dots (\mathbf{x}^M, G_x^M)\}$$

$$\mathbf{x}^m = [x_1, \dots, x_N]$$

$$\text{graph } G_x = (V_x, A_x)$$

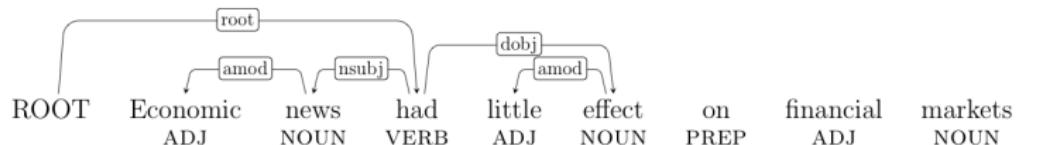
$$\text{vertices } V_x = \{0, 1, \dots, N\}$$

$$\text{edges } A_x = \{(i, j, k) | i, j \in V, k \in L(\text{labels})\}$$

$$\text{states } \mathbf{S}^m = [S_1, \dots, S_T]$$

$$\text{actions } \alpha^m = [\alpha_1, \dots, \alpha_T]$$

# Feature Representation



Stack = [ROOT, had, effect]

Buffer = [on, financial, markets, .]

What features would help us predict the correction action

Right-Arc(*prep*)?

# Feature Representation

- **Words/PoS in stack and buffer:**

wordS1=effect, wordB1=on, wordS2=had, posS1=NOUN,  
etc.

# Feature Representation

- **Words/PoS in stack and buffer:**

wordS1=effect, wordB1=on, wordS2=had, posS1=NOUN,  
etc.

- **Dependencies so far:**

depS1=dobj, depLeftChildS1=amod,  
depRightChildS1=NULL, etc.

# Feature Representation

- **Words/PoS in stack and buffer:**  
wordS1=effect, wordB1=on, wordS2=had, posS1=NOUN, etc.
- **Dependencies so far:**  
depS1=dobj, depLeftChildS1=amod,  
depRightChildS1=NULL, etc.
- **Previous actions:**  
 $\alpha_{t-1} = \text{Right-Arc}(dobj)$ , etc.

## Transition-based vs Graph-based parsing

- Transition-based tends to be better on shorter sentences, graph-based on longer ones

## Transition-based vs Graph-based parsing

- Transition-based tends to be better on shorter sentences, graph-based on longer ones
- Graph-based tends to be better on long-range dependencies

## Transition-based vs Graph-based parsing

- Transition-based tends to be better on shorter sentences, graph-based on longer ones
- Graph-based tends to be better on long-range dependencies
- Graph-based lacks the rich structural features

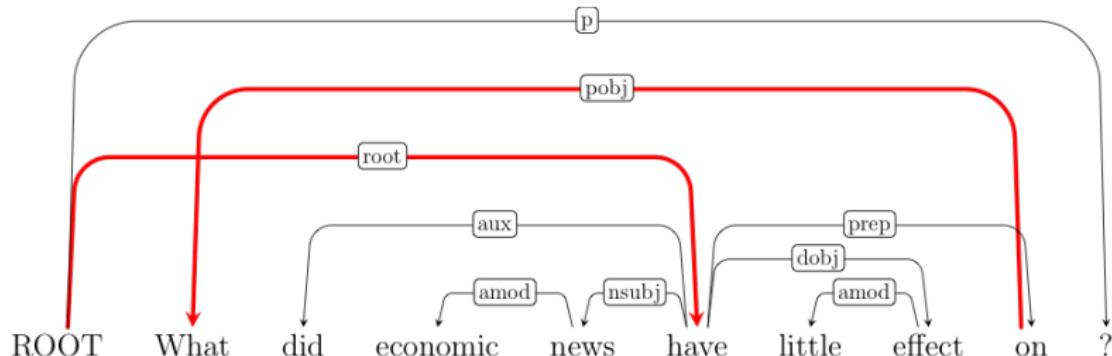
## Transition-based vs Graph-based parsing

- Transition-based tends to be better on shorter sentences, graph-based on longer ones
- Graph-based tends to be better on long-range dependencies
- Graph-based lacks the rich structural features
- Transition-based is greedy and suffers from early mistakes

## Transition-based vs Graph-based parsing

- Transition-based tends to be better on shorter sentences, graph-based on longer ones
- Graph-based tends to be better on long-range dependencies
- Graph-based lacks the rich structural features
- Transition-based is greedy and suffers from early mistakes
- Actually, can we ameliorate the greedy issue?  
Use Beam Search!

# Non-Projectivity



- Arcs are crossing each other
- long-range dependencies
- free word order

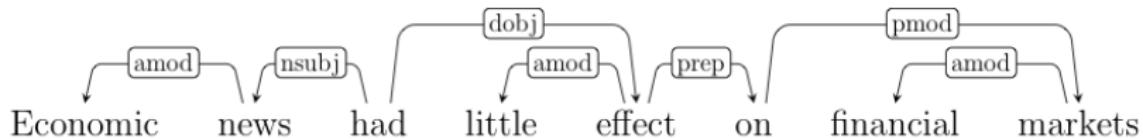
# Non-projective Transition-based parsing

- The standard stack-based systems cannot do it.
- But there are extensions:
  - swap actions: word reordering
  - k-planar parsing: use multiple stacks (usually 2)
- Standard graph-based parsing handles non-projectivity.

# Incremental Language Processing

- Other problems solved with similar approaches (a.k.a. transition-based, greedy):
  - semantic parsing (converting a natural language utterance to a logical form)
  - coreference resolution
- Whenever you have a problem with a very large space of outputs, worth considering

# Evaluation



- **Head-finding word-accuracy:**
  - unlabelled: % of words with the right head
  - labelled: % of words with the right head and label
- **Sentence accuracy:** % of sentences with correct graph

# Bibliography

- Chapter 11 from Eisenstein
- Nivre and McDonald's tutorial [slides](#)
- Nivre's [article](#) on deterministic transition-based dependency parsing
- Nivre and McDonald's [paper](#) comparing their approaches

# Coming up next week...

- Feed-forward Neural Networks
- Getting ready for Assignment 2!

# Text Classification with Logistic Regression

## COM6513 Natural Language Processing

Nikos Aletras

Computer Science Department

Week 3  
Spring 2022



The  
University  
Of  
Sheffield.

In the previous lecture...

# Text units

## Raw text:

*As far as I'm concerned, this is Lynch at his best. 'Lost Highway' is a dark, violent, surreal, beautiful, hallucinatory masterpiece. 10 out of 10 stars.*

- **word (token/term)**: a sequence of one or more characters excluding whitespaces. Sometimes it includes n-grams.
- **document (text sequence/snippet)**: sentence, paragraph, section, chapter, entire document, search query, social media post, transcribed utterance, pseudo-documents (e.g. all tweets posted by a user), etc.

## Documents: Document-Word Matrix (Bag-of-Words)

- A matrix  $X$ ,  $|D| \times |\mathcal{V}|$  where rows are documents in corpus  $D$ , and columns are vocabulary words in  $\mathcal{V}$ .
- For each document, count how many times words  $w \in \mathcal{V}$  appear in it.

## Documents: Document-Word Matrix (Bag-of-Words)

- A matrix  $X$ ,  $|D| \times |\mathcal{V}|$  where rows are documents in corpus  $D$ , and columns are vocabulary words in  $\mathcal{V}$ .
- For each document, count how many times words  $w \in \mathcal{V}$  appear in it.

	bad	good	great	terrible
Doc 1	14	1	0	5
Doc 2	2	5	3	0
Doc 3	0	2	5	0

- $X$  can also be obtained by adding all the one-hot vectors of the words in the documents and then transpose!

# Weighting the Document-Word Matrix: TF.IDF

- Penalise words appearing in many documents.
- Multiply word frequencies with their inverted document frequencies:

$$idf_w = \log_{10} \frac{N}{df_w}$$

where  $N$  is the number of documents in the corpus,  $df_w$  is document frequency of word  $w$

- To obtain:

$$x_{id} = tf_{id} \log_{10} \frac{N}{df_{id}}$$

- We can also squash the raw frequency, by using the  $\log_{10}$ .

## In this lecture...

- Our first NLP problem: **Text Classification**
- How to train and evaluate a **Logistic Regression** classifier for text classification

## In this lecture...

- Our first NLP problem: **Text Classification**
- How to train and evaluate a **Logistic Regression** classifier for text classification
- Get fully prepared for Assignment 1 (Deadline Mar 14)!

# Text classification

A very common problem in NLP:

Given a piece of **text**, assign a **label** from a predefined set of labels

# Text classification

A very common problem in NLP:

Given a piece of **text**, assign a **label** from a predefined set of labels

What could the labels be?

# Label Types

- positive vs negative (e.g. sentiment in reviews)
- topics (e.g. sports vs. politics)
- author name (author identification)
- pass or fail in essay grading
- supporting a stance or not
- any task with a finite set of classes!

## Label Types in a more abstract way...

- **Binary** (0 or 1), e.g. a movie review is positive or negative

## Label Types in a more abstract way...

- **Binary** (0 or 1), e.g. a movie review is positive or negative
- **Multi-class** (1 out of k classes), e.g. what is the topic of a news article (one out of sports, politics, business or technology)

## Label Types in a more abstract way...

- **Binary** (0 or 1), e.g. a movie review is positive or negative
- **Multi-class** (1 out of k classes), e.g. what is the topic of a news article (one out of sports, politics, business or technology)
- **Multi-label** (n out of k classes), e.g. what are the topics of a news article (zero or more out of sports, politics, business or technology)

## Label Types in a more abstract way...

- **Binary** (0 or 1), e.g. a movie review is positive or negative
- **Multi-class** (1 out of k classes), e.g. what is the topic of a news article (one out of sports, politics, business or technology)
- **Multi-label** (n out of k classes), e.g. what are the topics of a news article (zero or more out of sports, politics, business or technology)
- **Real number**, predict the average review score of a movie between 1 and 5 (formally called regression).

## Label Types in a more abstract way...

- **Binary** (0 or 1), e.g. a movie review is positive or negative
- **Multi-class** (1 out of k classes), e.g. what is the topic of a news article (one out of sports, politics, business or technology)
- **Multi-label** (n out of k classes), e.g. what are the topics of a news article (zero or more out of sports, politics, business or technology)
- **Real number**, predict the average review score of a movie between 1 and 5 (formally called regression).
- In this lecture, we focus only on binary and multi-class problems.

# Text Types

- news articles
- social media posts
- legal, biomedical text
- any type of text!

## A little test...

Given the following **tweets** labelled with **sentiment**:

Label	Tweet
negative	Very sad about Joe.
negative	No Sat off...Have to work 6 days a week.
negative	I'm a sad panda today.
positive	Such a beautiful satisfying day of shopping. Loves it.
positive	Who else is in a happy mood??
positive	Actually quite happy today.

## A little test...

Given the following **tweets** labelled with **sentiment**:

Label	Tweet
negative	Very sad about Joe.
negative	No Sat off...Have to work 6 days a week.
negative	I'm a sad panda today.
positive	Such a beautiful satisfying day of shopping. Loves it.
positive	Who else is in a happy mood??
positive	Actually quite happy today.

- What **features** should be **indicative of** positive/negative class?

## A little test...

Given the following **tweets** labelled with **sentiment**:

Label	Tweet
negative	Very sad about Joe.
negative	No Sat off...Have to work 6 days a week.
negative	I'm a sad panda today.
positive	Such a beautiful satisfying day of shopping. Loves it.
positive	Who else is in a happy mood??
positive	Actually quite happy today.

- What **features** should be **indicative of** positive/negative **class**?
- Would they generalize to unseen data?

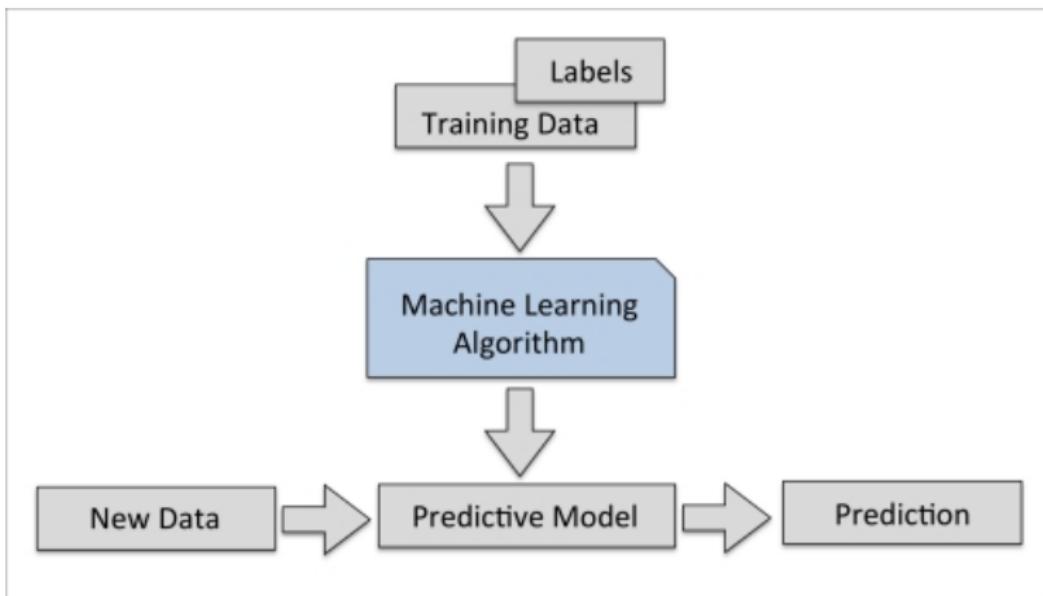
## A little test...

Given the following **tweets** labelled with **sentiment**:

Label	Tweet
negative	Very sad about Joe.
negative	No Sat off...Have to work 6 days a week.
negative	I'm a sad panda today.
positive	Such a beautiful satisfying day of shopping. Loves it.
positive	Who else is in a happy mood??
positive	Actually quite happy today.

- What **features** should be **indicative of** positive/negative **class**?
- Would they generalize to unseen data?
- Let's see how we can learn from data to predict class labels and class important features!

# Supervised Learning



## Supervised Learning - Exam Analogy

Imagine you want to prepare for an exam in a module.

## Supervised Learning - Exam Analogy

Imagine you want to prepare for an exam in a module.

- Your **training data** consists of only of all the available past exam papers.
- During training (studying), you learn by studying past exam papers.

# Supervised Learning - Exam Analogy

Imagine you want to prepare for an exam in a module.

- Your **training data** consists of only of all the available past exam papers.
- During training (studying), you learn by studying past exam papers.
- You can test yourself by holding out a number of past exams (**development/validation set**).

# Supervised Learning - Exam Analogy

Imagine you want to prepare for an exam in a module.

- Your **training data** consists of only of all the available past exam papers.
- During training (studying), you learn by studying past exam papers.
- You can test yourself by holding out a number of past exams (**development/validation set**).
- Evaluation is performed on the exam day (**test data**)! Your score is computed by your examiner.

# Supervised Learning

Given a set of  $M$  training pairs of documents  $\mathbf{x}$  (vectors!) and correct class labels  $y$ :

$$D_{train} = \{(\mathbf{x}^1, y^1) \dots (\mathbf{x}^M, y^M)\}$$

Learn a function (or model or classifier)  $f$  with parameters  $\mathbf{w}$  to predict the labels  $\hat{y}$  of any **new/unseen** document  $\mathbf{x}$  such that:

$$\hat{y} = f(\mathbf{x}, \mathbf{w})$$

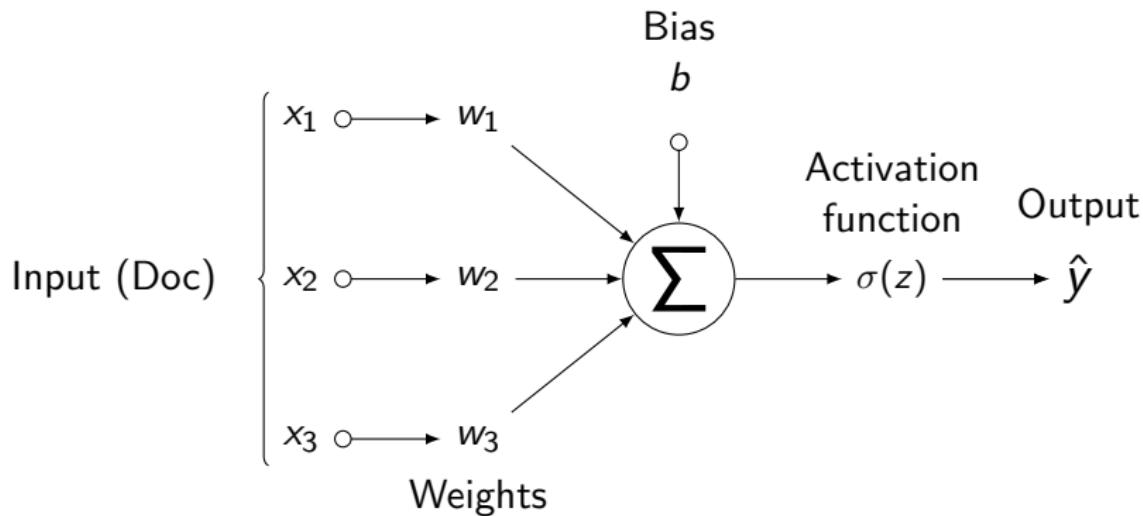
# Binary Logistic Regression

- Assume a document vector represented with counts over  $N$  words/features,  $\mathbf{x} \in \mathbb{R}^N$ .
- Our first classifier is a **linear model** where each element in  $\mathbf{x}$  is associated with a weight  $w_i$ , called **Logistic Regression**.  
**It's actually a classification algorithm!**

# Binary Logistic Regression

- Assume a document vector represented with counts over  $N$  words/features,  $\mathbf{x} \in \mathbb{R}^N$ .
- Our first classifier is a **linear model** where each element in  $\mathbf{x}$  is associated with a weight  $w_i$ , called **Logistic Regression**.  
*It's actually a classification algorithm!*
- How to predict the **class**  $\hat{y}$ , e.g. positive 1 or negative 0 sentiment, together with the **probability** for each class?

# Logistic Regression Overview



# Binary Logistic Regression

- Compute the dot product  $z$  between the input vector  $\mathbf{x}$  and the weight vector  $\mathbf{w}$ , and add a bias term  $b$  (often ignored):

$$z = \mathbf{w} \cdot \mathbf{x} + b$$

# Binary Logistic Regression

- Compute the dot product  $z$  between the input vector  $\mathbf{x}$  and the weight vector  $\mathbf{w}$ , and add a bias term  $b$  (often ignored):

$$z = \mathbf{w} \cdot \mathbf{x} + b$$

- Compute the probability of the positive class using the sigmoid function  $\sigma(\cdot)$ :

$$P(y = 1 | \mathbf{x}; \mathbf{w}) = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

# Binary Logistic Regression

- Compute the dot product  $z$  between the input vector  $\mathbf{x}$  and the weight vector  $\mathbf{w}$ , and add a bias term  $b$  (often ignored):

$$z = \mathbf{w} \cdot \mathbf{x} + b$$

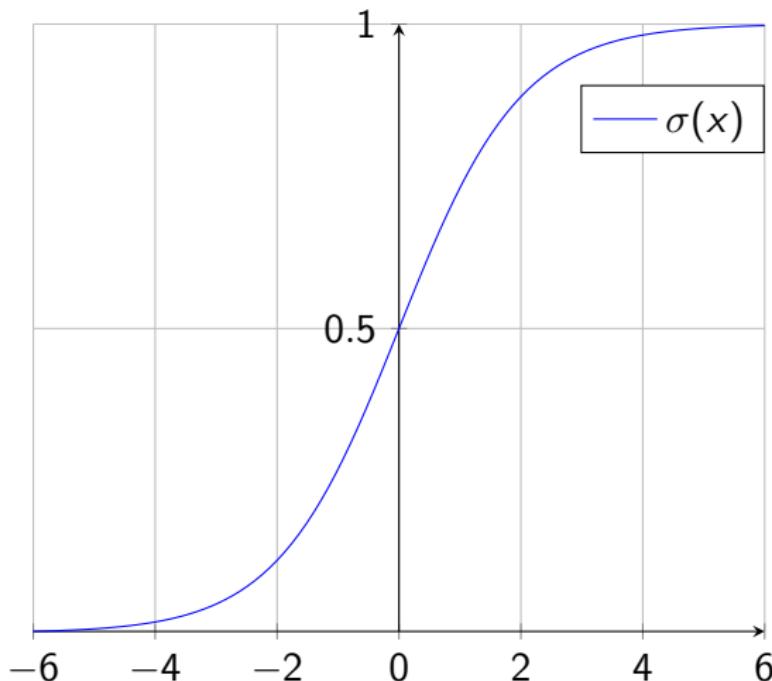
- Compute the probability of the positive class using the sigmoid function  $\sigma(\cdot)$ :

$$P(y = 1|\mathbf{x}; \mathbf{w}) = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

- Predict the class with the highest probability:

$$\hat{y} := \begin{cases} 0 & \text{if } P(y = 1|\mathbf{x}; \mathbf{w}) < 0.5 \\ 1 & \text{otherwise} \end{cases}$$

# Sigmoid function



Squashes a value  $x$  between 0 and 1 (or a vector, applied independently to each element).

# Multiclass Logistic Regression

- More than one labels:  $y \in \mathcal{Y} = \{0, \dots, k\}$ . E.g. is a news article about sports ( $y = 0$ ), politics ( $y = 1$ ) or technology ( $y = 2$ )

# Multiclass Logistic Regression

- More than one labels:  $y \in \mathcal{Y} = \{0, \dots, k\}$ . E.g. is a news article about sports ( $y = 0$ ), politics ( $y = 1$ ) or technology ( $y = 2$ )
- A matrix of weights  $W$ ,  $k \times n$  where  $k$  is the number of classes and  $n$  is the number of features in the input vector.
- Resulting into a vector of weights per class  $y$

# Multiclass Logistic Regression

- Compute the product between the input vector  $\mathbf{x}$  and the weight matrix  $W$ , and add a bias vector  $\mathbf{b} \in \mathbb{1}^k$  of ones (often ignored) to the resulting vector  $\mathbf{z}$ :

$$\mathbf{z} = W^T \mathbf{x} + \mathbf{b}$$

# Multiclass Logistic Regression

- Compute the product between the input vector  $\mathbf{x}$  and the weight matrix  $W$ , and add a bias vector  $\mathbf{b} \in \mathbb{1}^k$  of ones (often ignored) to the resulting vector  $\mathbf{z}$ :

$$\mathbf{z} = W^T \mathbf{x} + \mathbf{b}$$

- Compute the probability for each class  $c$  using the softmax function:

$$P(y = c|x; W) = \text{softmax}(\mathbf{z}) = \frac{\exp(z_c)}{\sum_{i \in \mathcal{Y}} \exp(z_i)}$$

# Multiclass Logistic Regression

- Compute the product between the input vector  $\mathbf{x}$  and the weight matrix  $W$ , and add a bias vector  $\mathbf{b} \in \mathbb{1}^k$  of ones (often ignored) to the resulting vector  $\mathbf{z}$ :

$$\mathbf{z} = W^T \mathbf{x} + \mathbf{b}$$

- Compute the probability for each class  $c$  using the softmax function:

$$P(y = c|x; W) = \text{softmax}(\mathbf{z}) = \frac{\exp(z_c)}{\sum_{i \in \mathcal{Y}} \exp(z_i)}$$

- Predict the class with the highest probability:

$$\hat{y} := \arg \max_c P(y = c|\mathbf{x}; W)$$

## Softmax function

Squeezes the values of a vector between 0 and 1 and the elements add up to 1 resulting into a probability distribution:

$$\begin{bmatrix} 1.2 \\ 0.9 \\ 0.4 \end{bmatrix} \rightarrow \text{softmax} \rightarrow \begin{bmatrix} 0.46 \\ 0.34 \\ 0.20 \end{bmatrix}$$

Wait...

How can we learn the weights?!

# Training

- We start by initialising  $\mathbf{w}$  with 0s.

# Training

- We start by initialising  $\mathbf{w}$  with 0s.
- We adjust  $\mathbf{w}$  by iterating over training data:

$$D_{train} = \{(x^1, y^1) \dots (x^M, y^M)\}$$

- But how can we decide how much we adjust the weights?

# Training

- We start by initialising  $\mathbf{w}$  with 0s.
- We adjust  $\mathbf{w}$  by iterating over training data:

$$D_{train} = \{(x^1, y^1) \dots (x^M, y^M)\}$$

- But how can we decide how much we adjust the weights?
- We need a function that tells us the difference between predicted and true labels (**loss or cost or objective function**)!

# Training

- We start by initialising  $\mathbf{w}$  with 0s.
- We adjust  $\mathbf{w}$  by iterating over training data:

$$D_{train} = \{(x^1, y^1) \dots (x^M, y^M)\}$$

- But how can we decide how much we adjust the weights?
- We need a function that tells us the difference between predicted and true labels (**loss or cost or objective function**)!
- So we can keep iterating over the training data and adjust the weights towards minimising the loss!

## Binary Cross-Entropy Loss

- We assume that the weights  $\mathbf{w}$  should maximise the log-likelihood of the correct class:

$$\log(P(y_i = c | \mathbf{x}_i; \mathbf{w}))$$

- Since we want to **minimise** a loss function, we take the negative of the log-likelihood:

$$L_{BCE} = -y \cdot \log(P(y = 1)) - (1 - y) \log(1 - P(y = 1))$$

- Cross-entropy loss increases as the predicted probability diverges from the actual label.
- Note that  $\log$  is a natural logarithm

## Categorical Cross-Entropy Loss

- To extend into multi-class, we just need to compute the negative log-likelihood of the true class  $y_c$ :

$$L_{CE} = -y_c \cdot \log(P(y_c = 1 | x_i; W_c))$$

- The loss of all other classes is 0
- $y_c$  is either 0 or 1,  $c$  takes values from 1, ..,  $k$  (number of classes)

## How to adjust the weights?

Numerical Optimisation is the research field that studies how to max/minimise the value of a function  $f(w)$  by changing  $w$ .

In our case (and a lot of supervised machine learning):

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} L(\mathbf{w}; x_i; y_i)$$

## A simpler case..

Binary logistic regression has one parameter per feature → many parameters!

Let's look at a simpler case:

$$x^* = \arg \min_{x \in \Re} f(x), f(x) = x^2$$

## Gradient-based optimisation

- How can we use the knowledge that  $f(x) = x^2$  in selecting points?

# Gradient-based optimisation

- How can we use the knowledge that  $f(x) = x^2$  in selecting points?
- Gradient of the function  $f$  w.r.t to parameter  $x$ :

$$\nabla_x f(x)$$

# Gradient-based optimisation

- How can we use the knowledge that  $f(x) = x^2$  in selecting points?
- Gradient of the function  $f$  w.r.t to parameter  $x$ :

$$\nabla_x f(x)$$

- When evaluated at  $x_k$ :
  - $\text{sign}(\nabla_x f(x))$  tells us if by increasing  $x$ ,  $f(x)$  will increase (+) or decrease (-)
  - $|\nabla_x f(x)|$  tells us how fast the in/decrease will be

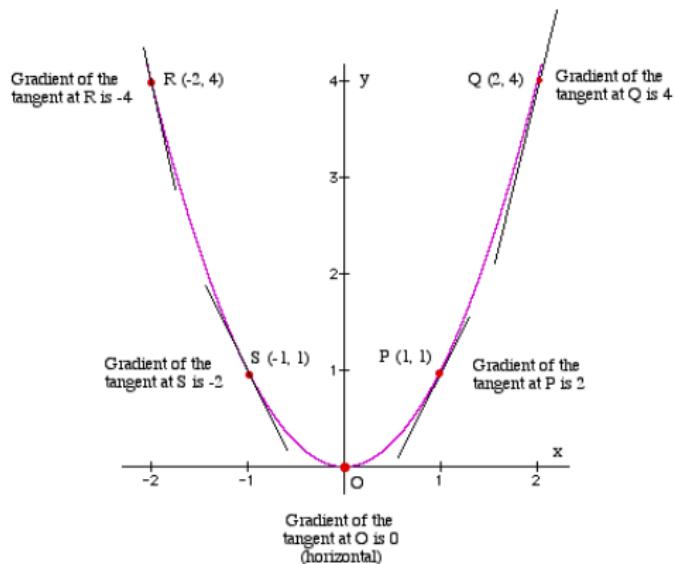
# Gradient-based optimisation

- How can we use the knowledge that  $f(x) = x^2$  in selecting points?
- Gradient of the function  $f$  w.r.t to parameter  $x$ :

$$\nabla_x f(x)$$

- When evaluated at  $x_k$ :
  - $\text{sign}(\nabla_x f(x))$  tells us if by increasing  $x$ ,  $f(x)$  will increase (+) or decrease (-)
  - $|\nabla_x f(x)|$  tells us how fast the in/decrease will be
- What does it mean if the gradient at  $x_k$  is 0?
  - Reached a minimum, no **single** direction to go

# Gradient-based optimisation



$$f(x) = x^2$$

$$\nabla_x f(x) = 2x$$

$f(x)$  is convex, thus if  $\nabla_x f(x_k) = 0$  then  $x_k = \arg \min_{x \in \Re} f(x)$

## Gradient of the Loss wrt to the weights

The gradient of the loss wrt to the parameter vector  $\mathbf{w} \in \Re^n$  is decomposed into the partial derivatives wrt to each parameter  $w_k$ :

$$\nabla_{\mathbf{w}} L(\mathbf{w}; \mathbf{x}_i; y_i) = \left( \frac{\partial \log P(y|x_i; \mathbf{w})}{\partial w_1}, \dots, \frac{\partial \log P(y|x_i; \mathbf{w})}{\partial w_n} \right)$$

$$\begin{aligned}\frac{\partial L(\mathbf{w}; \mathbf{x}_i; y_i)}{\partial w_j} &= \frac{\partial \log P(y|\mathbf{x}_i; \mathbf{w})}{\partial w_j} \\&= \dots \\&= (P(y|x_i; \mathbf{w}) - y_i) \cdot \mathbf{x}_i\end{aligned}$$

You can find more details on deriving the gradients [here](#).

# Stochastic Gradient Descent (SGD)

**Input:**  $D_{train} = \{(x_1, y_1) \dots (x_M, y_M)\}$ ,  $D_{val} = \{(x_1, y_1) \dots (x_D, y_D)\}$ ,  
*learning rate  $\eta$ , epochs  $e$ , tolerance  $t$*

*initialize  $\mathbf{w}$  with zeros*

**for each epoch  $e$  do**

*randomise order in  $D_{train}$*

**for each  $(x_i, y_i)$  in  $D_{train}$  do**

*update  $\mathbf{w} = \mathbf{w} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}; x_i; y_i)$*

*monitor training and validation loss*

*if previous validation loss – current validation loss; smaller than  $t$*

**break**

**return  $\mathbf{w}$**

$\eta$  denotes how much you want to update the weights  $\mathbf{w}$

## Stochastic Gradient Descent (SGD) for Multi-class

- You compute the gradient for the weights of the correct class only.
- Gradient is computed as in the binary case.

# Other Gradient Descent Optimisers

- **Gradient Descent:** computes the gradient of the loss function wrt to the parameters for the entire training set
- **Batch Gradient Descent:** computes the gradient of the loss function wrt to the parameters for small parts of the training set
- You can find more details [in this blog post](#).

# Regularisation

- Any model with a lot of features is prone to **overfitting** its training data: high training accuracy, low test accuracy.

# Regularisation

- Any model with a lot of features is prone to **overfitting** its training data: high training accuracy, low test accuracy.
- Avoid it by adding a regulariser in the objective:

$$L_{reg} = L + \alpha R(\mathbf{w})$$

# Regularisation

- Any model with a lot of features is prone to **overfitting** its training data: high training accuracy, low test accuracy.
- Avoid it by adding a regulariser in the objective:

$$L_{reg} = L + \alpha R(\mathbf{w})$$

- If  $R(\mathbf{w}) = \sum_{k=1}^K w_k^2$  ( $L_2$ -regularisation) then:

$$\frac{\partial L_{reg}(\mathbf{w}; D_{train})}{\partial w_k} = \frac{\partial L(\mathbf{w}; D_{train})}{\partial w_k} + 2\alpha w_k$$

- $\alpha$  is the regularisation strength

# Regularisation

- Any model with a lot of features is prone to **overfitting** its training data: high training accuracy, low test accuracy.
- Avoid it by adding a regulariser in the objective:

$$L_{reg} = L + \alpha R(\mathbf{w})$$

- If  $R(\mathbf{w}) = \sum_{k=1}^K w_k^2$  ( $L_2$ -regularisation) then:

$$\frac{\partial L_{reg}(\mathbf{w}; D_{train})}{\partial w_k} = \frac{\partial L(\mathbf{w}; D_{train})}{\partial w_k} + 2\alpha w_k$$

- $\alpha$  is the regularisation strength
- Intuitively: prefer small parameter values, by not updating as much.

# Other popular supervised ML algorithms

- Perceptron
- Support Vector Machines
- Naive Bayes
- Neural Networks (Weeks 6-10)
- Gaussian Processes

# Evaluation

- The standard way to evaluate a classifier is:

$$Accuracy = \frac{\# \text{correctly classified}}{\# \text{all documents}}$$

- What could go wrong?

# Evaluation

- The standard way to evaluate a classifier is:

$$Accuracy = \frac{\# \text{correctly classified}}{\# \text{all documents}}$$

- What could go wrong?
- When one class is much more common than the other, predicting it always gives high accuracy.

# Evaluation

Predicted/Correct	MinorityClass	MajorityClass
MinorityClass	TruePositive	FalsePositive
MajorityClass	FalseNegative	TrueNegative

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

$$F1\_Score = 2 \frac{Precision \times Recall}{Precision + Recall}$$

# Evaluation

Predicted/Correct	MinorityClass	MajorityClass
MinorityClass	TruePositive	FalsePositive
MajorityClass	FalseNegative	TrueNegative

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

$$F1\_Score = 2 \frac{Precision \times Recall}{Precision + Recall}$$

Use macro scores (averaging across classes) in multiclass classification (with imbalanced data).

# Bibliography

- Chapter 5 from Jurafsky & Martin.
- Sections 2.5 and 2.6 from Eisenstein.
- For more background reading on classification, Kevin Murphy's [introduction](#) touches upon most important concepts in ML.

## Coming up next week

- So far we saw how to do text classification using
  - a bag of words representation
  - and the logistic regression classifier
- But we have ignored word order. Language is structured! How we can develop sequential language models?

# Statistical Language Modelling

## COM6513 Natural Language Processing

Nikos Aletras

Computer Science Department

Week 4  
Spring 2022



The  
University  
Of  
Sheffield.

In the previous lecture...

- Our first NLP problem: **Text classification**

In the previous lecture...

- Our first NLP problem: **Text classification**
- But we ignored **word order** (apart from short sequences, e.g. n-grams)!

In this lecture...

Our second NLP problem: Language Modelling

What is the probability of a given sequence of words in a particular language (e.g. English)?

In this lecture...

Our second NLP problem: Language Modelling

What is the probability of a given sequence of words in a particular language (e.g. English)?

Odd problem. Applications?

# Applications of LMs

- Word likelihood for query completion in information retrieval (“Is Sheffield” → try it on your search engine)

# Applications of LMs

- Word likelihood for query completion in information retrieval (“Is Sheffield” → try it on your search engine)
- Language detection (“Ciao Sheffield” is it Italian or English?)

# Applications of LMs

- Word likelihood for query completion in information retrieval (“Is Sheffield” → try it on your search engine)
- Language detection (“Ciao Sheffield” is it Italian or English?)
- Grammatical error detection (“You’re welcome” or “Your welcome”?)

# Applications of LMs

- Word likelihood for query completion in information retrieval (“Is Sheffield” → try it on your search engine)
- Language detection (“Ciao Sheffield” is it Italian or English?)
- Grammatical error detection (“You’re welcome” or “Your welcome”?)
- Speech recognition (“I was tired too.” or “I was tired two.”?)

## Problem setup

Training data is a (often large) set of sentences  $\mathbf{x}^m$  with words  $x_n$ :

$$D_{train} = \{\mathbf{x}^1, \dots, \mathbf{x}^M\}$$

$$\mathbf{x} = [x_1, \dots, x_N]$$

for example:

$$\mathbf{x} = [<\text{s}>, \text{The}, \text{water}, \text{is}, \text{clear}, ., </\text{s}>]$$

$<\text{s}>$ : denotes start of the sentence

$</\text{s}>$ : denotes end of the sentence

# Calculate sentence probabilities

We want to learn a model that returns the **probability of an unseen sentence  $x$ :**

$$P(x) = P(x_1, \dots, x_n), \text{ for } \forall x \in V^{maxN}$$

$V$  is the vocabulary and  $V^{maxN}$  all possible sentences.

# Calculate sentence probabilities

We want to learn a model that returns the **probability of an unseen sentence  $x$ :**

$$P(x) = P(x_1, \dots, x_n), \text{ for } \forall x \in V^{maxN}$$

$V$  is the vocabulary and  $V^{maxN}$  all possible sentences.

How to compute probability?

## Unigram language model

Multiply the probability of each word appearing in the sentence  $\mathbf{x}$  computed over the entire corpus:

$$P(\mathbf{x}) = \prod_{n=1}^N P(x_n) = \prod_{n=1}^N \frac{c(x_n)}{\sum_{x \in V} c(x)}$$

# Unigram language model

Multiply the probability of each word appearing in the sentence  $\mathbf{x}$  computed over the entire corpus:

$$P(\mathbf{x}) = \prod_{n=1}^N P(x_n) = \prod_{n=1}^N \frac{c(x_n)}{\sum_{x \in V} c(x)}$$

```
< s > i love playing basketball < /s >
< s > arctic monkeys are from sheffield < /s >
< s > i study in sheffield uni < /s >
```

# Unigram language model

Multiply the probability of each word appearing in the sentence  $\mathbf{x}$  computed over the entire corpus:

$$P(\mathbf{x}) = \prod_{n=1}^N P(x_n) = \prod_{n=1}^N \frac{c(x_n)}{\sum_{x \in V} c(x)}$$

```
< s > i love playing basketball < /s >
< s > arctic monkeys are from sheffield < /s >
< s > i study in sheffield uni < /s >
```

$$P(i \text{ love}) = P(i)P(\text{love}) = \frac{2}{20} \cdot \frac{1}{20} = 0.005$$

# What could go wrong?

```
<s> i love playing basketball </s>
<s> arctic monkeys are from sheffield </s>
<s> i study in sheffield uni </s>
```

- The most probable word is  $\langle s \rangle$  ( $\frac{3}{20}$ )

# What could go wrong?

```
<s> i love playing basketball </s>  
<s> arctic monkeys are from sheffield </s>  
<s> i study in sheffield uni </s>
```

- The most probable word is  $\langle s \rangle$  ( $\frac{3}{20}$ )
- The most probable single-word sentence is “ $\langle s \rangle$ ”

# What could go wrong?

```
<s> i love playing basketball </s>  
<s> arctic monkeys are from sheffield </s>  
<s> i study in sheffield uni </s>
```

- The most probable word is  $\langle s \rangle$  ( $\frac{3}{20}$ )
- The most probable single-word sentence is “ $\langle s \rangle$ ”
- The most probable two-word sentence is “ $\langle s \rangle \langle s \rangle$ ”

# What could go wrong?

```
<s> i love playing basketball </s>  
<s> arctic monkeys are from sheffield </s>  
<s> i study in sheffield uni </s>
```

- The most probable word is  $\langle s \rangle$  ( $\frac{3}{20}$ )
- The most probable single-word sentence is “ $\langle s \rangle$ ”
- The most probable two-word sentence is “ $\langle s \rangle \langle s \rangle$ ”
- The most probable  $N$ -word sentence is  $N \times \langle s \rangle$

# Maximum Likelihood Estimation

Instead of assuming **independence**:

$$P(\mathbf{x}) = \prod_{n=1}^N P(x_n)$$

We assume that each word is **dependent** on all previous ones:

$$\begin{aligned} P(\mathbf{x}) &= P(x_1, \dots, x_N) \\ &= P(x_1)P(x_2 \dots x_N | x_1) \\ &= P(x_1)P(x_2 | x_1) \dots P(x_N | x_1, \dots, x_{N-1}) \\ &= \prod_{n=1}^N P(x_n | x_1, \dots, x_{n-1}) \quad (\text{chain rule}) \end{aligned}$$

# Maximum Likelihood Estimation

Instead of assuming **independence**:

$$P(\mathbf{x}) = \prod_{n=1}^N P(x_n)$$

We assume that each word is **dependent** on all previous ones:

$$\begin{aligned} P(\mathbf{x}) &= P(x_1, \dots, x_N) \\ &= P(x_1)P(x_2 \dots x_N | x_1) \\ &= P(x_1)P(x_2 | x_1) \dots P(x_N | x_1, \dots, x_{N-1}) \\ &= \prod_{n=1}^N P(x_n | x_1, \dots, x_{n-1}) \quad (\text{chain rule}) \end{aligned}$$

What could go wrong?

## Problems with MLE

Let's analyse this:

$$P(\mathbf{x}) = P(x_1)P(x_2|x_1)P(x_3|x_2, x_1)\dots P(x_N|x_1, \dots, x_{N-1})$$

$$P(x_n|x_{n-1}\dots x_1) = \frac{c(x_1\dots x_{n-1}, x_n)}{c(x_1\dots x_{n-1})}$$

As we condition on more words, the counts become **sparser**

# Bigram Language Models

Assume that the choice of a word **depends only on the one before it**:

$$P(\mathbf{x}) = \prod_{n=1}^N P(x_n|x_{n-1}) = \prod_{n=1}^N \frac{c(x_{n-1}, x_n)}{c(x_{n-1})}$$

k-th order **Markov assumption**:

$$P(x_n|x_{n-1}, \dots, x_1) \approx P(x_n|x_{n-1}, \dots, x_{n-k})$$

with k=1



## Bigram LM: From counts to probabilities

Unigram counts:

arctic	monkeys	are	my	favourite	band
100	600	4000	3000	500	200

Bigram counts (rows:  $x_{i-1}$ , cols:  $x_i$ ):

	arctic	monkeys	are	my	favourite	band
arctic	0	10	2	0	0	0
monkeys	0	0	250	1	5	0
are	3	45	0	600	25	1
my	0	2	0	1	300	5
favourite	0	1	0	0	0	50
band	0	0	3	10	0	0

## Bigram LM: From counts to probabilities

From the bigram count matrix, compute probabilities by dividing each cell by the appropriate unigram count for its row.

Bigram probabilities (rows:  $x_{i-1}$ , cols:  $x_i$ ):

	arctic	monkeys	are	my	favourite	band
arctic	0	0.1	0.02	0	0	0
monkeys	0	0	0.417	0.0017	0.008	0
are	0.0008	0.0113	0	0.15	0.0063	0.00003
my	0	0.0007	0	0.0003	0.1	0.0017
favourite	0	0.002	0	0	0	0.1
band	0	0	0.015	0.05	0	0

## Example: Bigram language model

$\mathbf{x} = [\text{arctic}, \text{monkeys}, \text{are}, \text{my}, \text{favourite}, \text{band}]$

$$\begin{aligned} P(\mathbf{x}) &= P(\text{monkeys}|\text{arctic})P(\text{are}|\text{monkeys})P(\text{my}|\text{are}) \\ &\quad P(\text{favourite}|\text{my})P(\text{band}|\text{favourite}) \\ &= \frac{c(\text{arctic,monkeys})}{c(\text{arctic})} \cdots \frac{c(\text{favourite,band})}{c(\text{favourite})} \\ &= 0.1 \cdot 0.417 \cdot 0.15 \cdot 0.1 \cdot 0.1 \\ &= 0.00006255 \end{aligned}$$

## Longer contexts (N-gram LMs)

$$P(x|context) = \frac{P(context, x)}{P(context)} = \frac{c(context, x)}{c(context)}$$

- In bigram LM *context* is  $x_{n-1}$ , trigram  $x_{n-2}, x_{n-1}$ , etc.
- The longer the context:
  - the more likely to capture long-range dependencies:  
“I saw a tiger that was really very...” fierce or talkative?
  - the sparser the counts (zero probabilities)
- 5-grams and training sets with billions of tokens are common.

## Unknown Words

- If a word was never encountered in training, any sentence containing it will have probability 0
- It happens:
  - all corpora are finite
  - new words emerging

# Unknown Words

- If a word was never encountered in training, any sentence containing it will have probability 0
- It happens:
  - all corpora are finite
  - new words emerging
- Common solutions:
  - Generate unknown words in the training data by replacing low-frequency words with a special UNKNOWN token
  - Use classes of unknown words, e.g. names and numbers

# Implementation details

- Dealing with large datasets requires efficiency:
  - use log probabilities to avoid underflows (small numbers)
  - efficient data structures for sparse counts, e.g. lossy data structures Bloom filters)

# Implementation details

- Dealing with large datasets requires efficiency:
  - use log probabilities to avoid underflows (small numbers)
  - efficient data structures for sparse counts, e.g. lossy data structures Bloom filters)

How do we train and evaluate our language models?

- We need train/dev/test data
- Evaluation approaches

## Intrinsic Evaluation: Accuracy

- How well does our LM predict the next word?
- **I always order pizza with cheese and...**
  - mushrooms?
  - bread?
  - and?
- Accuracy: how often the LM predicts the correct word
- The higher the better

## Intrinsic Evaluation: Perplexity

- **Perplexity:** the inverse probability of the test set

$\mathbf{x} = [x_1, \dots, x_N]$ , normalised by the number of words  $N$ :

$$PP(\mathbf{x}) = P(x_1, \dots, x_N)^{1/N}$$

$$= \sqrt[N]{\frac{1}{P(x_1, \dots, x_N)}}$$

$$= \sqrt[N]{\frac{1}{\prod_{i=1}^N P(x_i|x_1, \dots, x_{i-1})}}$$

## Intrinsic Evaluation: Perplexity

- **Perplexity:** the inverse probability of the test set

$\mathbf{x} = [x_1, \dots, x_N]$ , normalised by the number of words  $N$ :

$$\begin{aligned} PP(\mathbf{x}) &= P(x_1, \dots, x_N)^{1/N} \\ &= \sqrt[N]{\frac{1}{P(x_1, \dots, x_N)}} \\ &= \sqrt[N]{\frac{1}{\prod_{i=1}^N P(x_i|x_1, \dots, x_{i-1})}} \end{aligned}$$

- Measures how well a probability distribution predicts a sample.
- The lower the better.

Why is a bigram language model likely to have lower perplexity than a unigram one?

## Intrinsic Evaluation: Perplexity

- **Perplexity:** the inverse probability of the test set

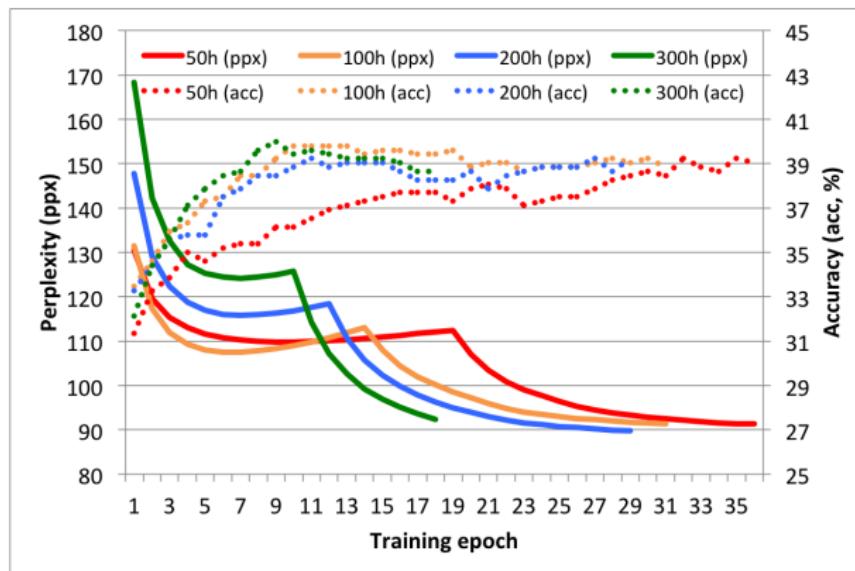
$\mathbf{x} = [x_1, \dots, x_N]$ , normalised by the number of words  $N$ :

$$\begin{aligned} PP(\mathbf{x}) &= P(x_1, \dots, x_N)^{1/N} \\ &= \sqrt[N]{\frac{1}{P(x_1, \dots, x_N)}} \\ &= \sqrt[N]{\frac{1}{\prod_{i=1}^N P(x_i|x_1, \dots, x_{i-1})}} \end{aligned}$$

- Measures how well a probability distribution predicts a sample.
- The lower the better.

Why is a bigram language model likely to have lower perplexity than a unigram one? **There is more context to predict the next word!**

# The problem with perplexity



- Doesn't always correlate with application performance
- Can't evaluate non probabilistic LMs

## Extrinsic Evaluation

- Sentence completion
- Grammatical error correction: detecting “odd” sentences and propose alternatives
- Natural language generation: prefer more “natural” sentences
- Speech recognition
- Machine translation

# Smoothing

- What happens when words that are in our vocabulary appear with an unseen context in test data?

# Smoothing

- What happens when words that are in our vocabulary appear with an unseen context in test data?
- They will be assigned with zero probability

# Smoothing

- What happens when words that are in our vocabulary appear with an unseen context in test data?
- They will be assigned with zero probability



**Smoothing (or discounting)** to the rescue: Steal from the rich and give to the poor!

# Smoothing intuition

- We often want to make estimates from sparse statistics:

$P(w | \text{denied the})$

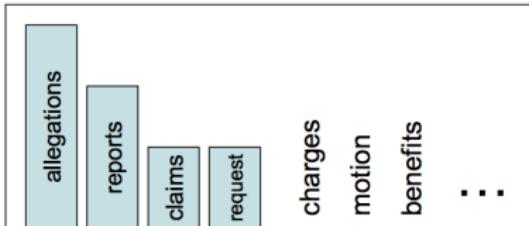
3 allegations

2 reports

1 claims

1 request

7 total



- Smoothing flattens spiky distributions so they generalize better

$P(w | \text{denied the})$

2.5 allegations

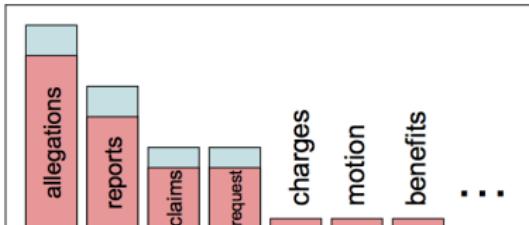
1.5 reports

0.5 claims

0.5 request

**2 other**

7 total



Taking from the frequent and giving to the rare (discounting)

## Add-1 Smoothing

Add-1 (or Laplace) smoothing adds one to all bigram counts:

$$P_{add-1}(x_n|x_{n-1}) = \frac{c(x_{n-1}, x_n) + 1}{c(x_{n-1}) + |V|}$$

Pretend we have seen all bigrams at least once!

## Add-k Smoothing

Add-1 puts too much probability mass to unseen bigrams, better to add- $k$ ,  $k < 1$ :

$$P_{add-k}(x_n|x_{n-1}) = \frac{counts(x_{n-1}, x_n) + k}{counts(x_{n-1}) + k|V|}$$

## Add-k Smoothing

Add-1 puts too much probability mass to unseen bigrams, better to add- $k$ ,  $k < 1$ :

$$P_{add-k}(x_n|x_{n-1}) = \frac{counts(x_{n-1}, x_n) + k}{counts(x_{n-1}) + k|V|}$$

$k$  is a hyperparameter: choose optimal value on the dev set!

# Interpolation

Longer contexts are more informative:

dog bites ... better than bites ...

but only if they are frequent enough:

canid bites ...better than bites ...?

# Interpolation

Longer contexts are more informative:

dog bites ... better than bites ...

but only if they are frequent enough:

canid bites ...better than bites ...?

Can we combine evidence from unigram, bigram and trigram probabilities?

# Simple Linear Interpolation

For a trigram LM:

$$\begin{aligned} P_{SLI}(x_n | x_{n-1}, x_{n-2}) &= \lambda_3 P(x_n | x_{n-1}, x_{n-2}) \\ &\quad + \lambda_2 P(x_n | x_{n-1}) \\ &\quad + \lambda_1 P(x_n) \quad \lambda_i > 0, \sum \lambda_i = 1 \end{aligned}$$

# Simple Linear Interpolation

For a trigram LM:

$$\begin{aligned} P_{SLI}(x_n | x_{n-1}, x_{n-2}) &= \lambda_3 P(x_n | x_{n-1}, x_{n-2}) \\ &\quad + \lambda_2 P(x_n | x_{n-1}) \\ &\quad + \lambda_1 P(x_n) \quad \lambda_i > 0, \sum \lambda_i = 1 \end{aligned}$$

- Weighted average of unigram, bigram and trigram probabilities

# Simple Linear Interpolation

For a trigram LM:

$$\begin{aligned} P_{SLI}(x_n | x_{n-1}, x_{n-2}) &= \lambda_3 P(x_n | x_{n-1}, x_{n-2}) \\ &\quad + \lambda_2 P(x_n | x_{n-1}) \\ &\quad + \lambda_1 P(x_n) \quad \lambda_i > 0, \sum \lambda_i = 1 \end{aligned}$$

- Weighted average of unigram, bigram and trigram probabilities
- How we choose the value of  $\lambda$ s?

# Simple Linear Interpolation

For a trigram LM:

$$\begin{aligned} P_{SLI}(x_n | x_{n-1}, x_{n-2}) &= \lambda_3 P(x_n | x_{n-1}, x_{n-2}) \\ &\quad + \lambda_2 P(x_n | x_{n-1}) \\ &\quad + \lambda_1 P(x_n) \quad \lambda_i > 0, \sum \lambda_i = 1 \end{aligned}$$

- Weighted average of unigram, bigram and trigram probabilities
- How we choose the value of  $\lambda$ s? Parameter tuning on the dev set!

## Backoff

Start with n-gram order of  $k$  but if the counts are 0 use  $k - 1$ :

$$BO(x_n | x_{n-1} \dots x_{n-k}) = \begin{cases} P(x_n | x_{n-1} \dots x_{n-k}), & \text{if } c(x_n \dots x_{n-k}) > 0 \\ BO(x_n | x_{n-1} \dots x_{n-k+1}), & \text{otherwise} \end{cases}$$

## Backoff

Start with n-gram order of  $k$  but if the counts are 0 use  $k - 1$ :

$$BO(x_n | x_{n-1} \dots x_{n-k}) = \begin{cases} P(x_n | x_{n-1} \dots x_{n-k}), & \text{if } c(x_n \dots x_{n-k}) > 0 \\ BO(x_n | x_{n-1} \dots x_{n-k+1}), & \text{otherwise} \end{cases}$$

Is this a probability distribution?

## Backoff

**NO!** Must discount probabilities for contexts with counts  $P^*$  and distribute the mass to the shorter context ones:

$$P_{BO}(x_n | x_{n-1} \dots x_{n-k}) = \begin{cases} P^*(x_n | x_{n-1} \dots x_{n-k}), & \text{if } c(x_n \dots x_{n-k}) > 0 \\ \alpha^{x_{n-1} \dots x_{n-k}} P_{BO}(x_n | x_{n-1} \dots x_{n-k+1}), & \text{otherwise} \end{cases}$$

$$\alpha^{x_{n-1} \dots x_{n-k}} = \frac{\beta^{x_{n-1} \dots x_{n-k}}}{\sum P_{BO}(x_n | x_{n-1} \dots x_{n-k+1})}$$

$\beta$ , is the left-over probability mass for the (n-k)-gram

# Absolute Discounting

Using 22M words for train and held-out

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Can you predict the heldout (test) set average count given the training?

# Absolute Discounting

Using 22M words for train and held-out

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Can you predict the heldout (test) set average count given the training?

Testing counts = training counts - 0.75 (absolute discount)

# Absolute discounting

$$P_{AbsDiscount}(x_n|x_{n-1}) = \frac{c(x_n, x_{n-1}) - d}{c(x_{n-1})} + \lambda_{x_{n-1}} P(x_n)$$

- $d = 0.75$ ,  $\lambda$ s tuned to ensure we have a valid probability distribution.
- Component of the **Kneser-Ney** discounting:
  - Intuition: a word can be very frequent, but if only follows very few contexts,  
e.g. **Francisco** is frequent but almost always follows **San**
  - The unigram probability in the context of the bigram should capture how likely  $x_n$  is to be a novel continuation.

## Stupid Backoff

- Do we really need probabilities? Estimating the additional parameters takes time for large corpora.

# Stupid Backoff

- Do we really need probabilities? Estimating the additional parameters takes time for large corpora.
- If scoring is enough, **stupid backoff** works adequately:

$$SBO(x_n | x_{n-1} \dots x_{n-k}) = \begin{cases} P(x_n | x_{n-1} \dots x_{n-k}), & \text{if } c(x_n \dots x_{n-k}) > 0 \\ \lambda SBO(x_n | x_{n-1} \dots x_{n-k+1}), & \text{otherwise} \end{cases}$$

- Empirically found that  $\lambda = 0.4$  works well

# Stupid Backoff

- Do we really need probabilities? Estimating the additional parameters takes time for large corpora.
- If scoring is enough, **stupid backoff** works adequately:

$$SBO(x_n | x_{n-1} \dots x_{n-k}) = \begin{cases} P(x_n | x_{n-1} \dots x_{n-k}), & \text{if } c(x_n \dots x_{n-k}) > 0 \\ \lambda SBO(x_n | x_{n-1} \dots x_{n-k+1}), & \text{otherwise} \end{cases}$$

- Empirically found that  $\lambda = 0.4$  works well
- They called it stupid because they didn't expect it to work well!

## Last words: More data defeats smarter models!

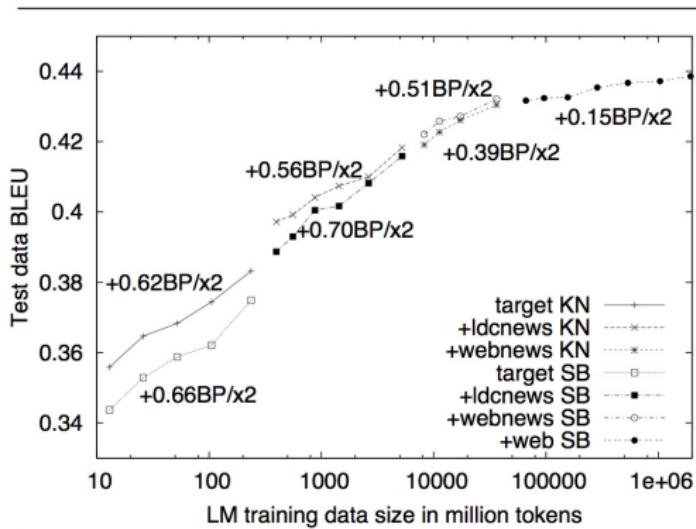


Figure 5: BLEU scores for varying amounts of data using Kneser-Ney (KN) and Stupid Backoff (SB).

From Large Language Models in Machine Translation

# Bibliography

- Chapter 3 from Jurafsky & Martin
- Chapter 6 from Eisentein
- Michael Collins' notes on LMs

## Coming up next...

- We have learned how to model word sequences using Markov models
- In the following lecture we will look at how to perform part-of-speech tagging using:
  - the **Hidden** Markov Model (HMM)
  - the **Conditional Random Fields** (CRFs), an extension of logistic regression for sequence modelling

# Sequence Labelling and Part-of-Speech Tagging

## COM6513 Natural Language Processing

Nikos Aletras

Computer Science Department

Week 5  
Spring 2022



The  
University  
Of  
Sheffield.

In the previous lecture...

- Our first **sequence modelling** problem: **Language Modelling**

In this lecture...

- What about if we want to **assign a label to each word in a sequence?**

## In this lecture...

- What about if we want to **assign a label to each word in a sequence?**
- Sequence labelling!

## In this lecture...

- What about if we want to **assign a label to each word in a sequence?**
- Sequence labelling!
- Applications?

# Applications

- Part-of-Speech (POS) Tagging

$(\mathbf{x}, \mathbf{y}) = ([I, studied, in, Sheffield],$   
 $[Pronoun, Verb, Preposition, ProperNoun])$

# Applications

- Part-of-Speech (POS) Tagging

$$(\mathbf{x}, \mathbf{y}) = ([I, studied, in, Sheffield], \\ [Pronoun, Verb, Preposition, ProperNoun])$$

- Named Entity Recognition

$$(\mathbf{x}, \mathbf{y}) = ([Giannis, Antetokounmpo, plays, for, the, Bucks], \\ [Person, Person, NotEnt, NotEnt, NotEnt, Org])$$

- Machine Translation (reconstruct word alignments)

$$(\mathbf{x}, \mathbf{y}) = ([la, maison, bleu], \\ [the, house, blue])$$

We will use POS tagging as a running example

# Parts of Speech (POS)

Label words according to their syntactic function in a sentence:

The	results	appear	in	today	's	news
determiner	noun	verb	preposition	noun	possessive	noun

# Parts of Speech (POS)

Label words according to their syntactic function in a sentence:

The	results	appear	in	today	's	news
determiner	noun	verb	preposition	noun	possessive	noun

What could they be useful for?

# Parts of Speech (POS)

Label words according to their syntactic function in a sentence:

The	results	appear	in	today	's	news
determiner	noun	verb	preposition	noun	possessive	noun

What could they be useful for?

- text classification
- language modelling
- syntactic parsing
- named entity recognition
- question answering

# PoS Tags

- Open class:  
nouns, verbs, adjectives
- Closed class:  
determiners, prepositions, conjunctions, etc

## PoS definitions

- Most research uses the Penn Treebank PoS tag set
- Includes 45 tags making distinctions between:
  - verbs in active vs past tense
  - nouns in singular vs plural number
  - etc.

# PoS definitions

- Most research uses the [Penn Treebank PoS tag set](#)
- Includes 45 tags making distinctions between:
  - verbs in active vs past tense
  - nouns in singular vs plural number
  - etc.
- Penn Tree Bank inspired by English. Recent work has focused on the [Universal PoS tag set](#):
  - 17 coarse tags: one noun class, one verb class, etc.
  - developed considering 22 languages

## Sequence labelling: Problem Setup

Data consists of word sequences with label sequences:

$$D_{train} = \{(\mathbf{x}^1, \mathbf{y}^1) \dots (\mathbf{x}^M, \mathbf{y}^M)\}$$

$$\mathbf{x}^m = [x_1, \dots, x_N]$$

$$\mathbf{y}^m = [y_1, \dots, y_N]$$

Learn a model  $f$  that predicts the best label sequence:

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} f(\mathbf{x}, \mathbf{y}) \quad (1)$$

$\mathbf{y} \in \mathcal{Y}^N$  is the set of all possible combinations of label sequences and  $\mathcal{Y} = \{A, B, C, \dots\}$  are the possible classes for each word.

# Could we use a dictionary-based model?

$\{the : \text{determiner}, can : \text{modal}, fly : \text{verb}\}$

# Could we use a dictionary-based model?

{*the* : determiner, *can* : modal, *fly* : verb}

Yes, but the same word can have different tags in different contexts.

I	can	fly
pronoun	modal	verb

vs:

I	can	fly
pronoun	verb	noun

**can** and 11.5% of the words in the Brown corpus have more than one tag

# Can we use a Markov model?

Use tags  $\mathbf{y}$  instead of words:

$$P(\mathbf{y}) = \prod_{n=1}^N P(y_n | y_{n-1})$$

# Can we use a Markov model?

Use tags **y** instead of words:

$$P(\mathbf{y}) = \prod_{n=1}^N P(y_n | y_{n-1})$$

Our training data should be able to tell us that tagging is unlikely:

I	can	fly
pronoun	modal	noun

# Can we use a Markov model?

Use tags  $\mathbf{y}$  instead of words:

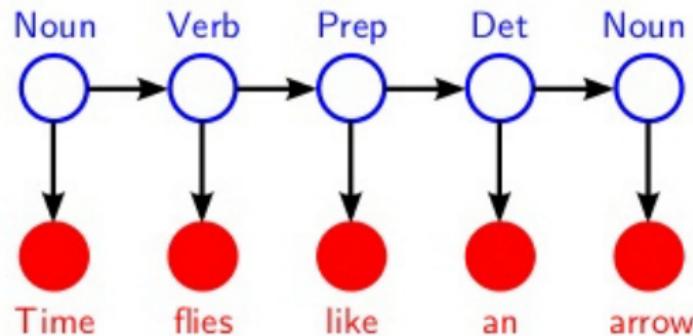
$$P(\mathbf{y}) = \prod_{n=1}^N P(y_n | y_{n-1})$$

Our training data should be able to tell us that tagging is unlikely:

I	can	fly
pronoun	modal	noun

What about the words? We will get the same  $N$ -tag long sequence for any sentence!

# Hidden Markov Model (HMM)



- Labels  $y_i$  (i.e. PoS tags) are hidden states emitting words.
- Assumptions:
  - 1st order Markov among the POS tags (current tag depends only on previous tag)
  - Each word only depends on its POS tag

# HMM: Derivation

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} P(\mathbf{y} | \mathbf{x}) \quad (\text{Bayes rule})$$

# HMM: Derivation

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} P(\mathbf{y} | \mathbf{x}) \quad (\text{Bayes rule})$$

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} \frac{P(\mathbf{x} | \mathbf{y}) P(\mathbf{y})}{P(\mathbf{x})} \quad (\text{word probabilities are constant})$$

# HMM: Derivation

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} P(\mathbf{y}|\mathbf{x}) \quad (\text{Bayes rule})$$

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} \frac{P(\mathbf{x}|\mathbf{y})P(\mathbf{y})}{P(\mathbf{x})} \quad (\text{word probabilities are constant})$$

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} P(\mathbf{x}|\mathbf{y})P(\mathbf{y}) \quad (\text{1st order Markov})$$

# HMM: Derivation

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} P(\mathbf{y} | \mathbf{x}) \quad (\text{Bayes rule})$$

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} \frac{P(\mathbf{x}|\mathbf{y})P(\mathbf{y})}{P(\mathbf{x})} \quad (\text{word probabilities are constant})$$

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} P(\mathbf{x}|\mathbf{y})P(\mathbf{y}) \quad (\text{1st order Markov})$$

$$\hat{\mathbf{y}} \approx \arg \max_{\mathbf{y} \in \mathcal{Y}^N} \prod_{n=1}^N P(x_n|y_n)P(y_n|y_{n-1})$$

# HMM: Training

- Maximum likelihood estimation (i.e. counts!):

$$P(y_n | y_{n-1}) = \frac{c(y_n, y_{n-1})}{c(y_{n-1})} \quad (\text{transition probabilities})$$

$$P(x_n | y_n) = \frac{c(x_n, y_n)}{c(y_n)} \quad (\text{emission probabilities})$$

# HMM: Training

- Maximum likelihood estimation (i.e. counts!):

$$P(y_n | y_{n-1}) = \frac{c(y_n, y_{n-1})}{c(y_{n-1})} \quad (\text{transition probabilities})$$

$$P(x_n | y_n) = \frac{c(x_n, y_n)}{c(y_n)} \quad (\text{emission probabilities})$$

- We can easily compute counts  $c(\cdot)$  using a labelled corpus (pairs of words-POS tags).

## HMM: Example

$x = [\text{START}, \text{I}, \text{can}, \text{fly}, \text{END}]$

$y = [\text{START}, \text{PPSS}, \text{MD}, \text{NN}, \text{END}]$

$$P(y|x) = P(I|PPSS)P(PPSS|START)$$

$$P(can|MD)P(MD|PPSS)$$

$$P(fly|NN)P(NN|MD)$$

# Decoding/Inference

- So we have everything we need to decode/infer the most likely tag sequence for a sentence:

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} \prod_{n=1}^N P(x_n | y_n) P(y_n | y_{n-1})$$

- Just enumerate all possible tag sequences?

# Decoding/Inference

- So we have everything we need to decode/infer the most likely tag sequence for a sentence:

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} \prod_{n=1}^N P(x_n | y_n) P(y_n | y_{n-1})$$

- Just enumerate all possible tag sequences?  
Intractable. We would need to evaluate  $|\mathcal{Y}|^N$  sequences!

# Decoding/Inference

- So we have everything we need to decode/infer the most likely tag sequence for a sentence:

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} \prod_{n=1}^N P(x_n | y_n) P(y_n | y_{n-1})$$

- Just enumerate all possible tag sequences?  
Intractable. We would need to evaluate  $|\mathcal{Y}|^N$  sequences!
- We will see later how to decode efficiently!

## HMMs: Some extra points

- Higher order HMMs:
  - longer contexts, more expensive inference
  - benefits are usually small

## HMMs: Some extra points

- Higher order HMMs:
  - longer contexts, more expensive inference
  - benefits are usually small
- Smoothing:
  - what happens when we have unseen word/tags or tag-tag combinations?

# HMMs: Some extra points

- Higher order HMMs:
  - longer contexts, more expensive inference
  - benefits are usually small
- Smoothing:
  - what happens when we have unseen word/tags or tag-tag combinations?

Use methods we learned in the language modeling lecture!

## HMMs: Limitations

- They generate probabilities for words and labels, we just want labels
- No overlapping features (e.g. unigrams+bigrams)
- No subword features (e.g. suffixes)

# Conditional Random Fields: Extend LR for Sequence Labelling

- Logistic regression can also provide probabilities but supports more flexible representations!

# Conditional Random Fields: Extend LR for Sequence Labelling

- Logistic regression can also provide probabilities but supports more flexible representations!
- Given a word, a candidate label of the current and the label of the previous word, predict the most likely label for that word using a (multi-class LR) in each time step → Conditional Random Fields

# Conditional Random Fields: Extend LR for Sequence Labelling

- Logistic regression can also provide probabilities but supports more flexible representations!
- Given a word, a candidate label of the current and the label of the previous word, predict the most likely label for that word using a (multi-class LR) in each time step → Conditional Random Fields

# Conditional Random Fields: Extend LR for Sequence Labelling

- Logistic regression can also provide probabilities but supports more flexible representations!
- Given a word, a candidate label of the current and the label of the previous word, predict the most likely label for that word using a (multi-class LR) in each time step → Conditional Random Fields
- CRF paper more than 11K citations since 2001, 10 year test of time award at ICML conference

# Conditional Random Fields

Decompose the per sentence  $\mathbf{x} = [x_1, \dots, x_N]$  prediction:

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} f(\mathbf{x}, \mathbf{y})$$

into each word  $x_n$ :

$$\hat{y}_n = \arg \max_{y \in \mathcal{Y}} f(x_n; y_{n-1}, n) = \arg \max_{y \in \mathcal{Y}} \mathbf{w}^y \phi(x_n, y_{n-1}, n)$$

# Conditional Random Fields

Decompose the per sentence  $\mathbf{x} = [x_1, \dots, x_N]$  prediction:

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} f(\mathbf{x}, \mathbf{y})$$

into each word  $x_n$ :

$$\hat{y}_n = \arg \max_{y \in \mathcal{Y}} f(x_n; y_{n-1}, n) = \arg \max_{y \in \mathcal{Y}} \mathbf{w}^y \phi(x_n, y_{n-1}, n)$$

- How to construct a feature vector  $\phi(x_n, y_n, y_{n-1}, n)$ ?

## CRF: Feature Vectors

- $\phi_1(x_n, y_n, y_{n-1}, n) = 1$  if  $y_n = ADVERB$  and the n-th word ends in “-ly”; 0 otherwise.  
“usually”, “casually”
- $\phi_2(x_n, y_n, y_{n-1}, n) = 1$  if  $n = 1$ ,  $y_n = VERB$ , and the sentence ends in a question mark; 0 otherwise.  
“Is it true?”
- etc.

## CRF: Inference

The normalisation factor has to score all possible label sequences for all sentences, so it is ignored:

$$\arg \max_{\mathbf{y} \in \mathcal{Y}^N} P_{CRF}(\mathbf{y} | \mathbf{x}; \mathbf{w}) = \arg \max_{\mathbf{y} \in \mathcal{Y}^N} \sum_{n=1}^N \mathbf{w} \cdot \phi(y_n, y_{n-1}, \mathbf{x}, n)$$

## CRF: Training

- Training by minimising the negative log-likelihood objective:

$$\mathbf{w} = \arg \min_{\mathbf{w} \in \Re^d} \sum_{m=1}^M -\log P_{CRF}(\mathbf{y}^m | \mathbf{x}^m; \mathbf{w})$$

- using **Stochastic Gradient Descent**

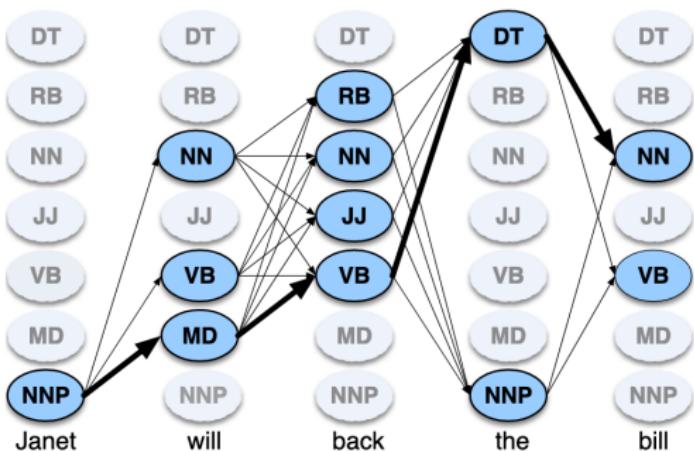
## Decoding with Viterbi

- Enumerating all possible tag sequences in HMM and CRF is intractable!

# Decoding with Viterbi

- Enumerating all possible tag sequences in HMM and CRF is intractable!
- Dynamic programming: store and re-use calculations
- Possible due to independence assumptions
- Keep track of the highest probability to reach each PoS tag for each word and how we got there

# Decoding with Viterbi



## Viterbi: Data structures

- Viterbi score matrix  $V^{|Y| \times N}$ :

# Viterbi: Data structures

- Viterbi score matrix  $V^{|\mathcal{Y}| \times N}$ :
  - Tag set  $\mathcal{Y}$ , sentence  $\mathbf{x} = [x_1, \dots, x_N]$

# Viterbi: Data structures

- **Viterbi score matrix**  $V^{|\mathcal{Y}| \times N}$ :
  - Tag set  $\mathcal{Y}$ , sentence  $\mathbf{x} = [x_1, \dots, x_N]$
  - each cell contains the highest prob. for word  $n$  with tag  $y$

# Viterbi: Data structures

- **Viterbi score matrix**  $V^{|\mathcal{Y}| \times N}$ :
  - Tag set  $\mathcal{Y}$ , sentence  $\mathbf{x} = [x_1, \dots, x_N]$
  - each cell contains the highest prob. for word  $n$  with tag  $y$
  - 1st order Markov: only depends on the previous tag  $y_{n-1}$ 
$$V[y, n] = \max_{y_{n-1} \in \mathcal{Y}} V[y_{n-1}, n-1] \times P(y_n | x_n, y_{n-1})$$

## Viterbi: Data structures

- **Backpointer matrix**  $backptr^{|\mathcal{Y}| \times N}$ :

# Viterbi: Data structures

- **Backpointer matrix**  $backptr^{|\mathcal{Y}| \times N}$ :
  - instead of the max score, keep the previous tag that got it

# Viterbi: Data structures

- **Backpointer matrix**  $backptr^{|\mathcal{Y}| \times N}$ :

- instead of the max score, keep the previous tag that got it
- $\text{argmax}$  instead of  $\max$

$$backptr[y, n] = \arg \max_{y' \in \mathcal{Y}} V[y', n - 1] \times P(y|y') \times P(x_n|y)$$

# Viterbi algorithm

**Input:** word sequence  $\mathbf{x} = [x_1, \dots, x_N]$ ,

$P(y_n|x_n, y_{n-1})$  probs

set matrix  $V^{|\mathcal{Y}| \times N} = 1$

**for**  $n = 1$  **to**  $N$  **do**

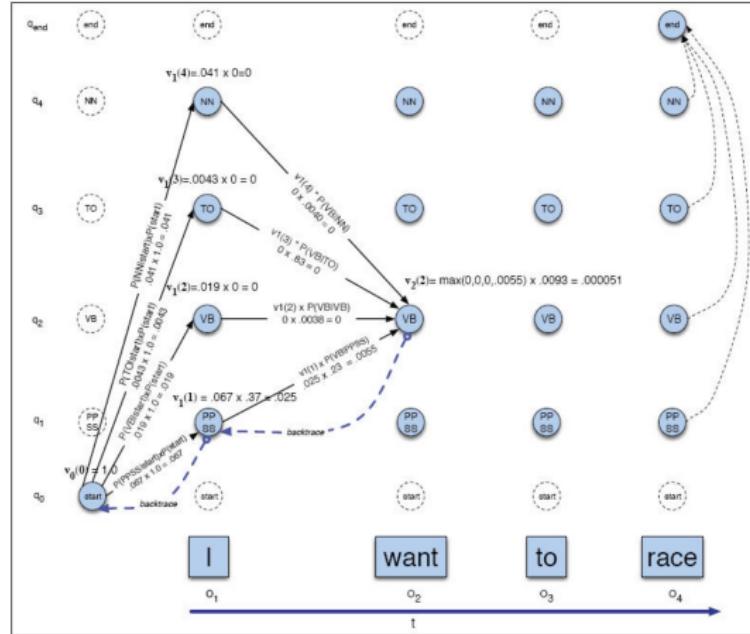
**for**  $y \in \mathcal{Y}$  **do**

$$V[y, n] = \max_{y_{n-1} \in \mathcal{Y}} V[y_{n-1}, n-1] \times P(y_n|x_n, y_{n-1})$$

$$backptr[y, n] = \arg \max_{y_{n-1} \in \mathcal{Y}} V[y_{n-1}, n-1] \times P(y_n|x_n, y_{n-1})$$

$$backptr[None, N+1] = \arg \max_{y_{n-1} \in \mathcal{Y}} V[y_{n-1}, N] \times P(None|y_{n-1})$$

# Viterbi diagram



Break the large arg max into smaller ones, left-to-right (**dynamic programming**)

## Beam Search: Inexact Decoding

- Viterbi performs exact search (under assumptions) by evaluating all options.

## Beam Search: Inexact Decoding

- Viterbi performs exact search (under assumptions) by evaluating all options.
- Get faster by being inexact, i.e. avoid labelling some candidate sequences with **Beam Search**

## Beam Search

- Do Viterbi, but keep only best  $k$  hypotheses at each step

## Beam Search

- Do Viterbi, but keep only best  $k$  hypotheses at each step
- If beam size is 1, then we have greedy search

# Beam Search

- Do Viterbi, but keep only best  $k$  hypotheses at each step
- If beam size is 1, then we have greedy search
- Often beams less than 10 get close to exact search, but much faster

# Beam Search

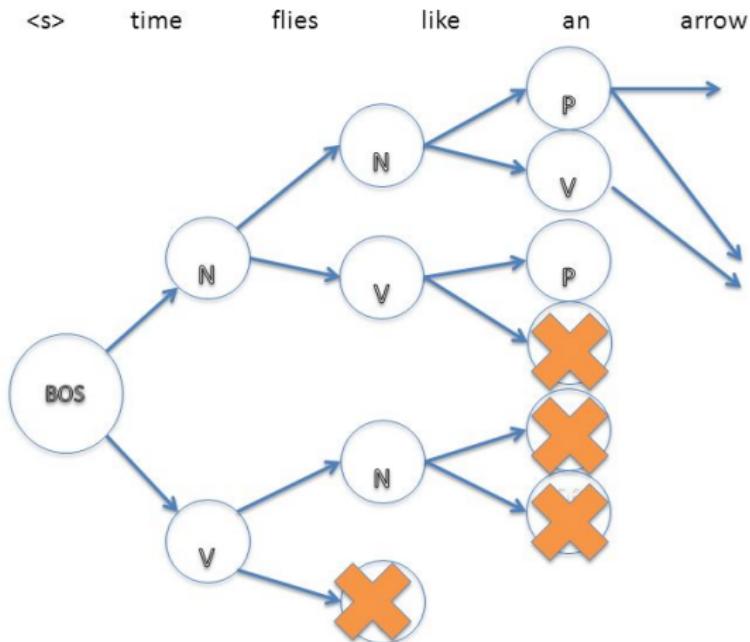
- Do Viterbi, but keep only best  $k$  hypotheses at each step
- If beam size is 1, then we have greedy search
- Often beams less than 10 get close to exact search, but much faster
- Beams must be of the same length to be comparable

# Beam Search

- Do Viterbi, but keep only best  $k$  hypotheses at each step
- If beam size is 1, then we have greedy search
- Often beams less than 10 get close to exact search, but much faster
- Beams must be of the same length to be comparable
- Attractive when we need complex feature functions i.e. avoid Markov assumptions)

## Beam Search: Example

## Beam Search, k=3



# Beam Search: Algorithm

```
Input: word sequence  $\mathbf{x} = [x_1, \dots, x_N]$ , weights  $\mathbf{w}$ 
set beam  $B = \{(\mathbf{y}_{\text{temp}} = [\text{START}], \text{score} = 0)\}$ , size  $k$ 
for  $n = 1$  to  $N$  do
     $B' = \{\}$ 
    for  $b \in B$  do
        for  $y \in \mathcal{Y}$  do
             $B' = B' \cup ([b.\mathbf{y}_{\text{temp}}; y], P([b.\mathbf{y}_{\text{temp}}; y] | x_n))$ 
     $B = \text{TOP-k}(B')$ 
return  $\text{TOP-1}(B)$ 
```

# Bibliography

- Chapter 8 from Jurafsky and Martin
- Sections 7.1-7.4, 7.5.3 and Chapter 8 from Eisenstein
- This blog post on CRFs by Edwin Chen
- Tutorial on CRFs by Sutton and McCallum

## Coming up next...

The best-studied, more complex than sequence labeling problem in NLP: **dependency parsing**

# Feedforward Neural Networkss: Revisiting Word Vectors and Text Classification

## COM6513 Natural Language Processing

Nikos Aletras

[n.aletras@sheffield.ac.uk](mailto:n.aletras@sheffield.ac.uk)

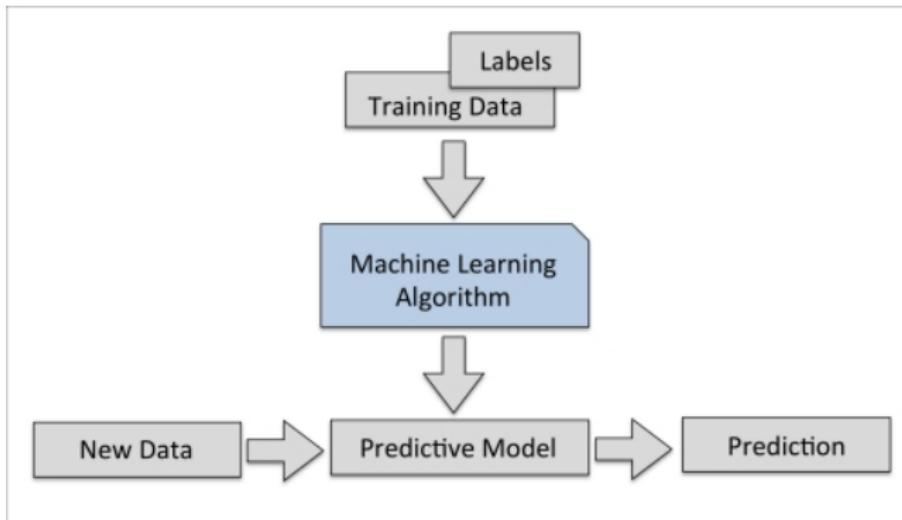
Computer Science Department

Week 6  
Spring 2022



The  
University  
Of  
Sheffield.

# In lecture 2...



Supervised ML

## In lecture 2...

- **Machine Learning Algorithm:** Logistic Regression
- Binary and Multi-class

## Logistic Regression recap

- Compute the dot product  $z$  between the input vector  $\mathbf{x}$  and the weight vector  $\mathbf{w}$ , and add a bias term  $b$  (often ignored):

$$z = \mathbf{w} \cdot \mathbf{x} + b$$

# Logistic Regression recap

- Compute the dot product  $z$  between the input vector  $\mathbf{x}$  and the weight vector  $\mathbf{w}$ , and add a bias term  $b$  (often ignored):

$$z = \mathbf{w} \cdot \mathbf{x} + b$$

- Compute the probability of the positive class using the sigmoid function  $\sigma(\cdot)$ :

$$P(y = 1 | \mathbf{x}; \mathbf{w}) = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

# Logistic Regression recap

- Compute the dot product  $z$  between the input vector  $\mathbf{x}$  and the weight vector  $\mathbf{w}$ , and add a bias term  $b$  (often ignored):

$$z = \mathbf{w} \cdot \mathbf{x} + b$$

- Compute the probability of the positive class using the sigmoid function  $\sigma(\cdot)$ :

$$P(y = 1 | \mathbf{x}; \mathbf{w}) = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

- Predict the class with the highest probability:

$$\hat{y} := \begin{cases} 0 & \text{if } P(y = 1 | \mathbf{x}; \mathbf{w}) < 0.5 \\ 1 & \text{otherwise} \end{cases}$$

## Logistic Regression recap

- Extend to multi-class by introducing **weights for each class** and use **softmax** instead of sigmoid

# Logistic Regression recap

- Extend to multi-class by introducing **weights for each class** and use **softmax** instead of sigmoid
- Learn the weights by minimising the **cross-entropy loss** using **Stochastic Gradient Descent**

# Logistic Regression recap

- Extend to multi-class by introducing **weights for each class** and use **softmax** instead of sigmoid
- Learn the weights by minimising the **cross-entropy loss** using **Stochastic Gradient Descent**
- LR directly maps input to output and only captures linear relationships in the data

In this lecture...

- **Feedforward neural networks** or deep feedforward networks or multilayer perceptrons

## In this lecture...

- **Feedforward neural networks** or deep feedforward networks or multilayer perceptrons
- Pass input through a series of intermediate computations (**hidden layers**) to capture **non-linear relationships** a.k.a. **deep learning**

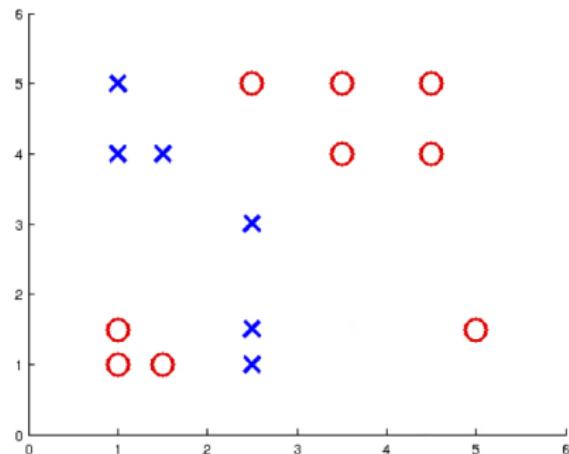
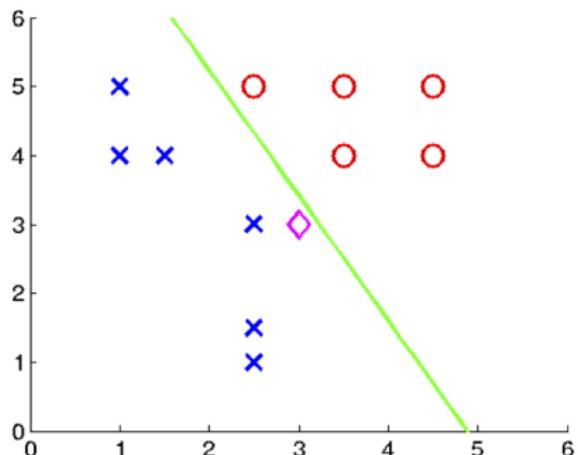
## In this lecture...

- **Feedforward neural networks** or deep feedforward networks or multilayer perceptrons
- Pass input through a series of intermediate computations (**hidden layers**) to capture **non-linear relationships** a.k.a. **deep learning**
- Train with **SGD** and **Backpropagation** (for computing the gradients)

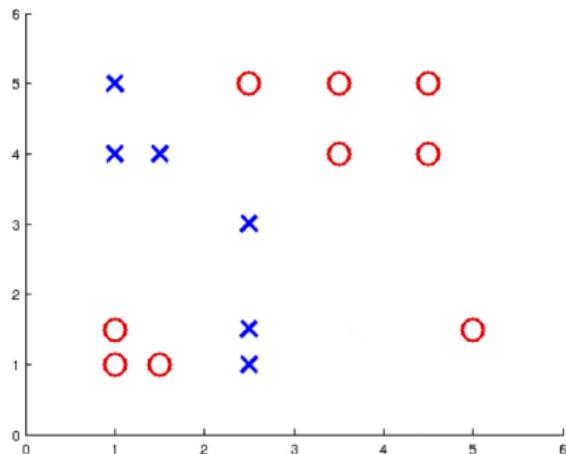
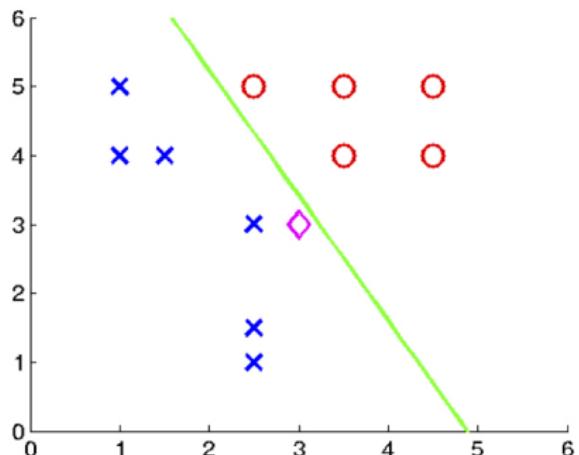
## In this lecture...

- **Feedforward neural networks** or deep feedforward networks or multilayer perceptrons
- Pass input through a series of intermediate computations (**hidden layers**) to capture **non-linear relationships** a.k.a. **deep learning**
- Train with **SGD** and **Backpropagation** (for computing the gradients)
- NLP applications: word vectors and text classification

# Limitations of linear models



# Limitations of linear models



The righthand dataset is not linearly separable and cannot be learned with a linear model.

# Feedforward Neural Network

- A feedforward network defines a mapping  $\mathbf{y} = f(\mathbf{x}; \mathbf{w})$  between an input  $\mathbf{x}$  and output  $\mathbf{y}$  given parameters  $\mathbf{w}$ .

# Feedforward Neural Network

- A feedforward network defines a mapping  $\mathbf{y} = f(\mathbf{x}; \mathbf{w})$  between an input  $\mathbf{x}$  and output  $\mathbf{y}$  given parameters  $\mathbf{w}$ .
- Feedforward nets compose together many different **functions** connected in a **chain**:  $f(\mathbf{x}) = f_3(f_2(f_1(\mathbf{x})))$

# Feedforward Neural Network

- A feedforward network defines a mapping  $\mathbf{y} = f(\mathbf{x}; \mathbf{w})$  between an input  $\mathbf{x}$  and output  $\mathbf{y}$  given parameters  $\mathbf{w}$ .
- Feedforward nets compose together many different **functions** connected in a **chain**:  $f(\mathbf{x}) = f_3(f_2(f_1(\mathbf{x})))$
- $f_1$  is the first **hidden layer** of the model,  $f_2$  the second and so on. Number of hidden layers denote the **depth** of the model.

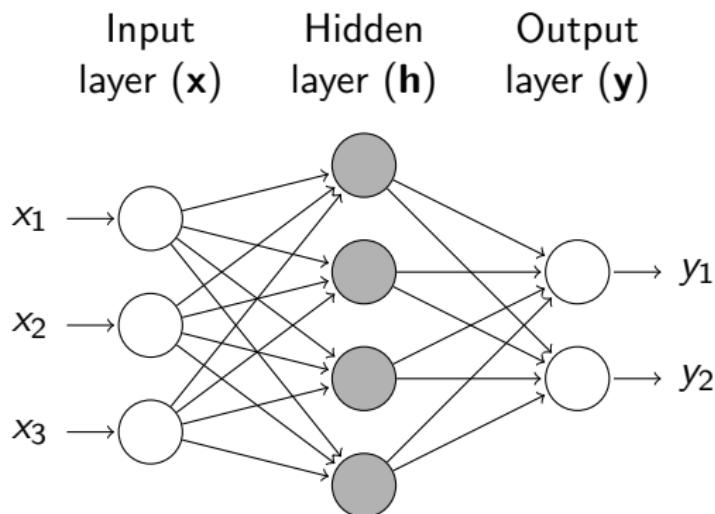
# Feedforward Neural Network

- A feedforward network defines a mapping  $\mathbf{y} = f(\mathbf{x}; \mathbf{w})$  between an input  $\mathbf{x}$  and output  $\mathbf{y}$  given parameters  $\mathbf{w}$ .
- Feedforward nets compose together many different **functions** connected in a **chain**:  $f(\mathbf{x}) = f_3(f_2(f_1(\mathbf{x})))$
- $f_1$  is the first **hidden layer** of the model,  $f_2$  the second and so on. Number of hidden layers denote the **depth** of the model.
- **Input** denote the input layer

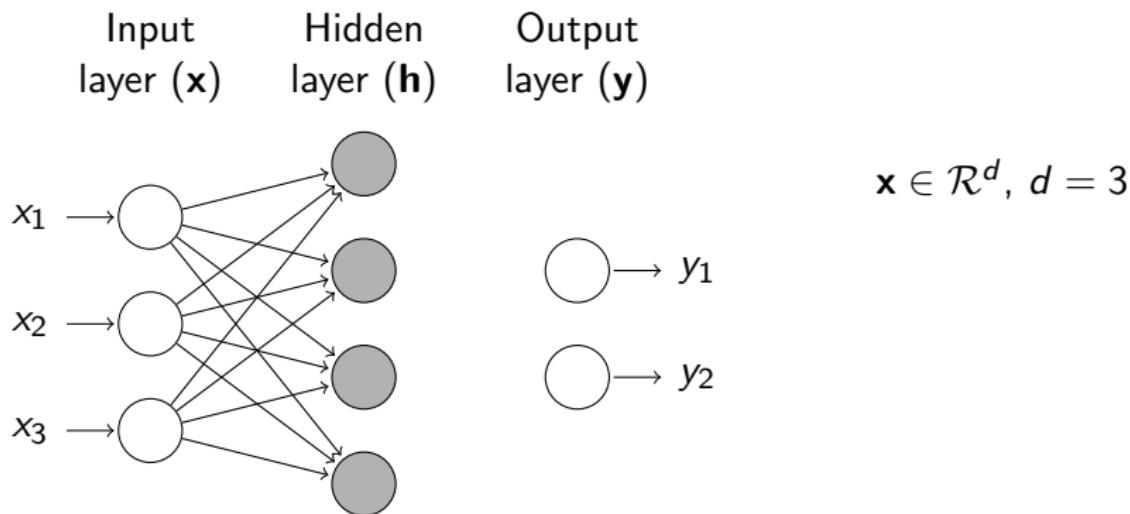
# Feedforward Neural Network

- A feedforward network defines a mapping  $\mathbf{y} = f(\mathbf{x}; \mathbf{w})$  between an input  $\mathbf{x}$  and output  $\mathbf{y}$  given parameters  $\mathbf{w}$ .
- Feedforward nets compose together many different **functions** connected in a **chain**:  $f(\mathbf{x}) = f_3(f_2(f_1(\mathbf{x})))$
- $f_1$  is the first **hidden layer** of the model,  $f_2$  the second and so on. Number of hidden layers denote the **depth** of the model.
- **Input** denote the input layer
- The final layer to obtain the prediction is called the **output** layer (e.g. sigmoid, softmax)

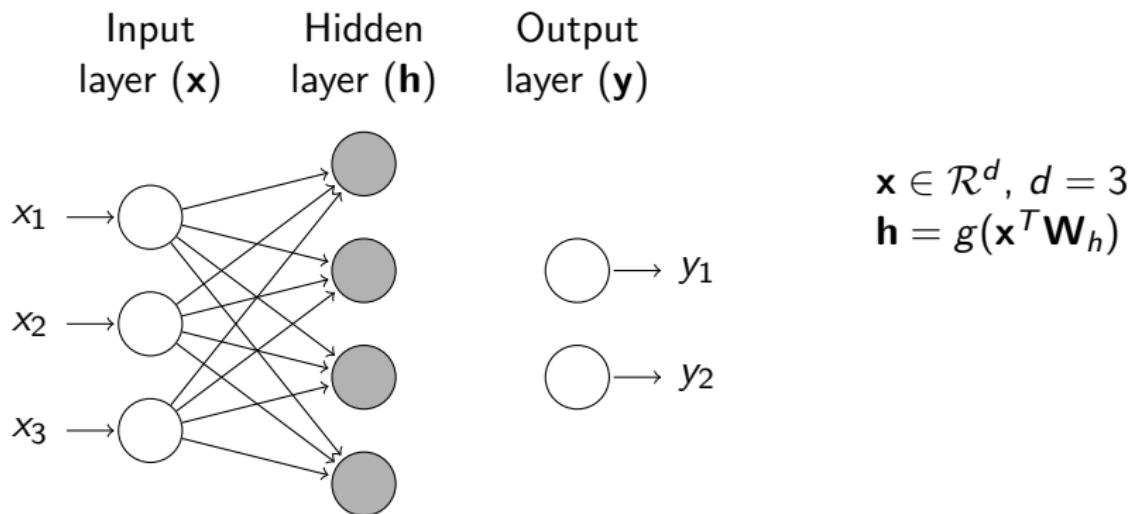
# Feedforward Neural Network



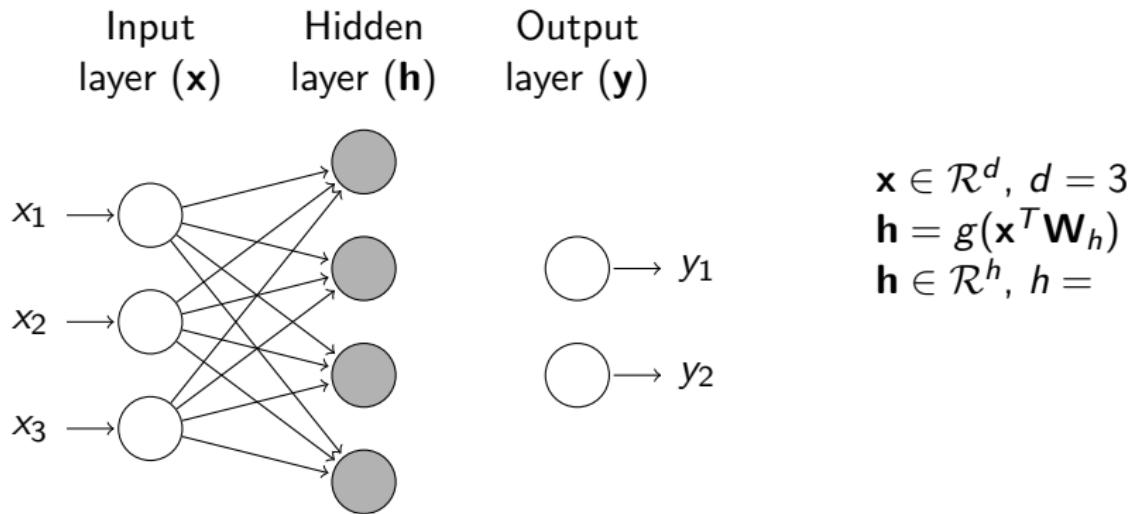
# Feedforward Neural Network



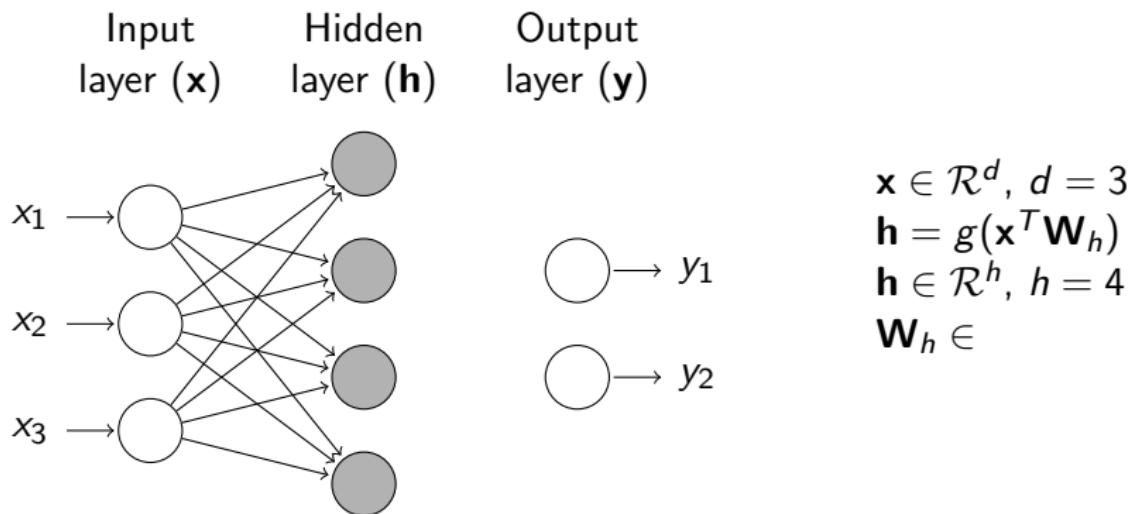
# Feedforward Neural Network



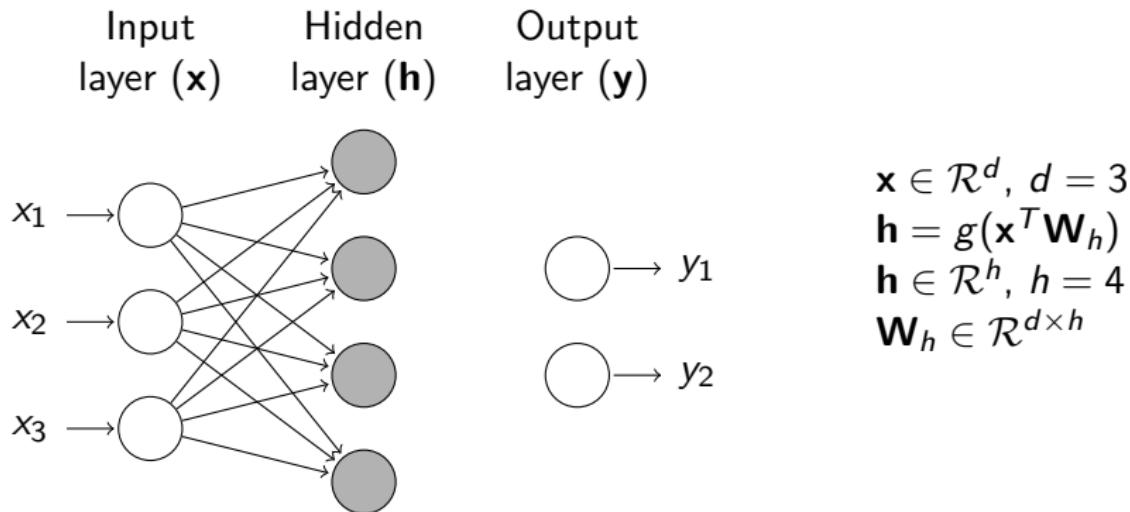
# Feedforward Neural Network



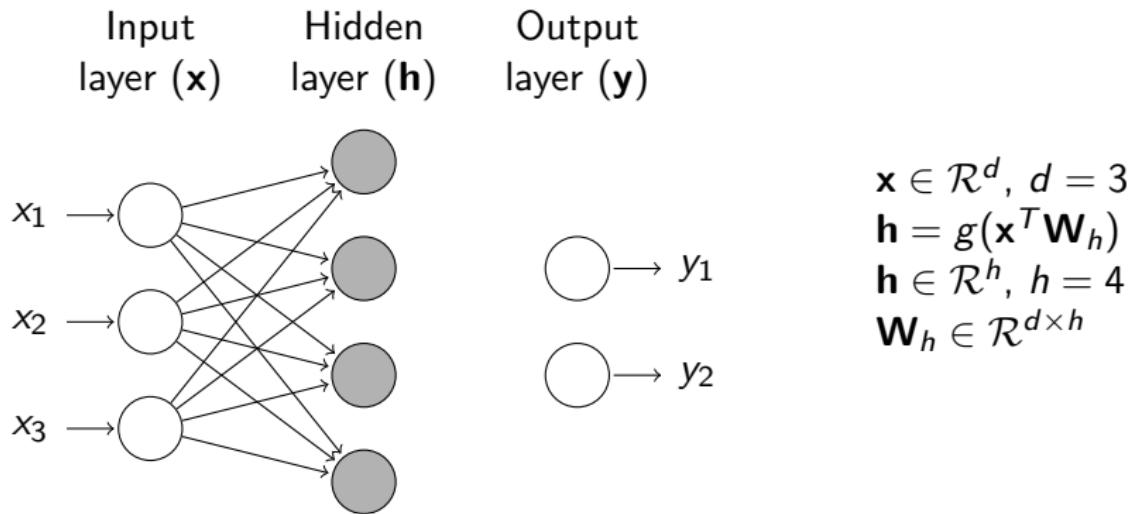
# Feedforward Neural Network



# Feedforward Neural Network



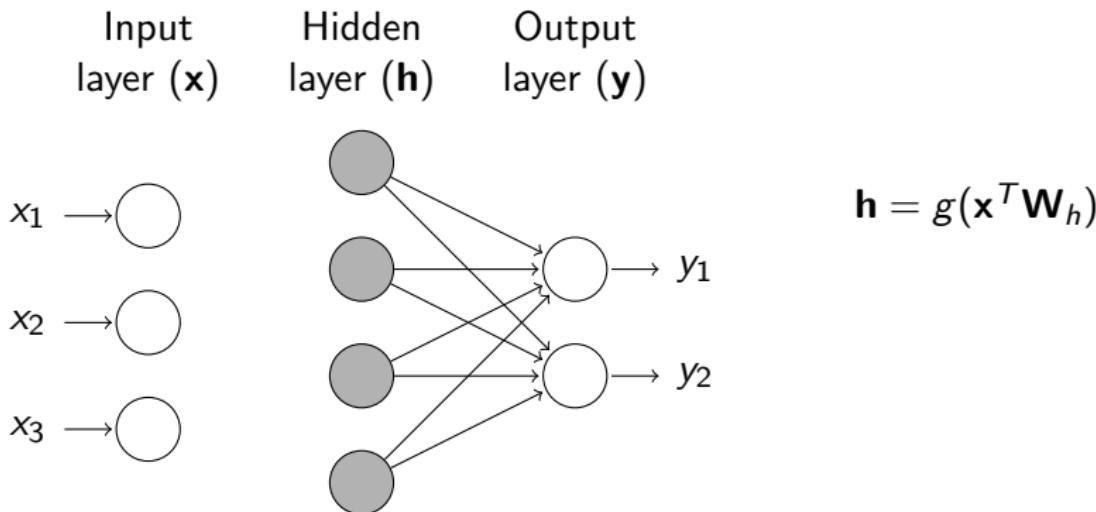
# Feedforward Neural Network



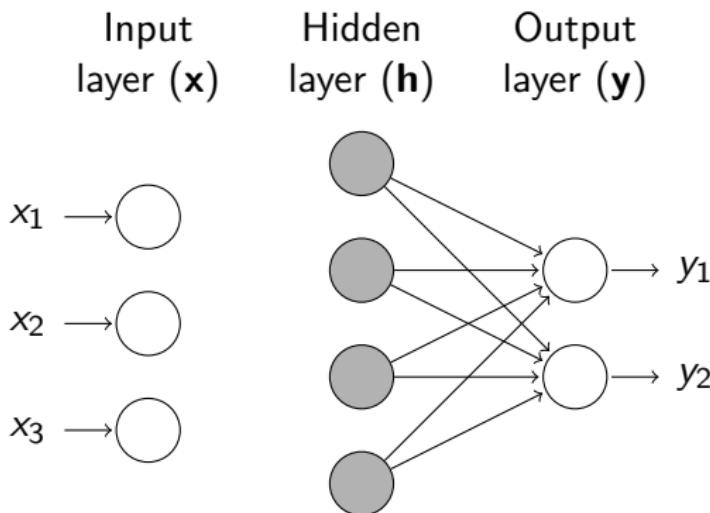
Extended to deeper architectures:

$$\mathbf{h}_i = g(\mathbf{h}_{i-1}^T \mathbf{W}_{h_i})$$

# Feedforward Neural Network

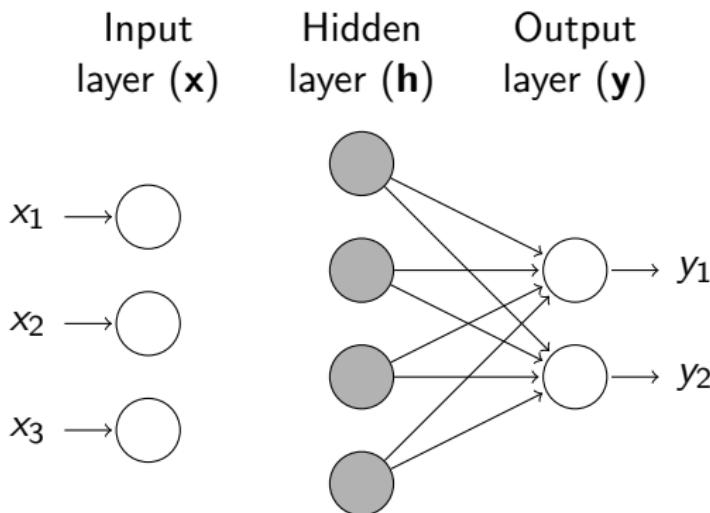


# Feedforward Neural Network



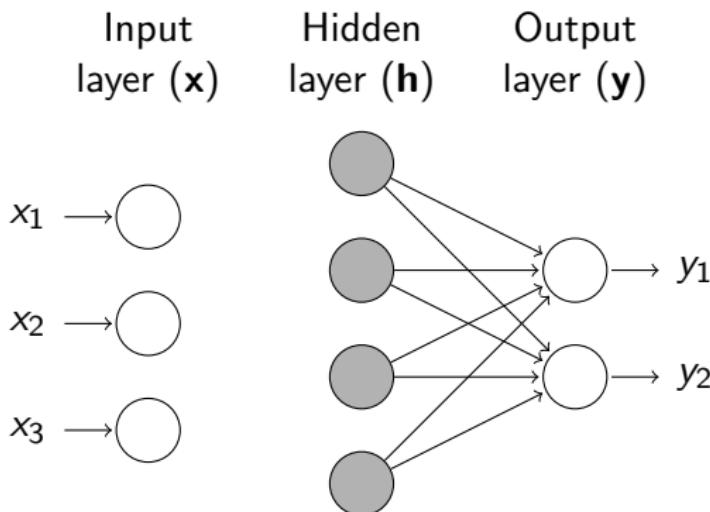
$$\begin{aligned}\mathbf{h} &= g(\mathbf{x}^T \mathbf{W}_h) \\ \mathbf{y} &= \text{softmax}(\mathbf{h}^T \mathbf{W}_o) \\ \mathbf{W}_o &\in\end{aligned}$$

# Feedforward Neural Network



$$\begin{aligned}\mathbf{h} &= g(\mathbf{x}^T \mathbf{W}_h) \\ \mathbf{y} &= \text{softmax}(\mathbf{h}^T \mathbf{W}_o) \\ \mathbf{W}_o &\in \mathcal{R}^{h \times y}\end{aligned}$$

# Feedforward Neural Network



$$\begin{aligned}\mathbf{h} &= g(\mathbf{x}^T \mathbf{W}_h) \\ \mathbf{y} &= \text{softmax}(\mathbf{h}^T \mathbf{W}_o) \\ \mathbf{W}_o &\in \mathcal{R}^{h \times y}\end{aligned}$$

But what is  $g(\cdot)$ ?

# Activation Functions

- Applied on **hidden units**: elements of  $h$

# Activation Functions

- Applied on **hidden units**: elements of  $h$
- Sigmoid:

$$g(z) = \sigma(z)$$

# Activation Functions

- Applied on **hidden units**: elements of  $h$
- **Sigmoid**:

$$g(z) = \sigma(z)$$

- **Hyperbolic Tangent**:

$$g(z) = \tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$

# Activation Functions

- Applied on **hidden units**: elements of  $h$
- **Sigmoid**:

$$g(z) = \sigma(z)$$

- **Hyperbolic Tangent**:

$$g(z) = \tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$

- **Rectified Linear Unit (ReLU)**:

$$g(z) = \max(0, z)$$

# Activation Functions

- Applied on **hidden units**: elements of  $h$
- **Sigmoid**:

$$g(z) = \sigma(z)$$

- **Hyperbolic Tangent**:

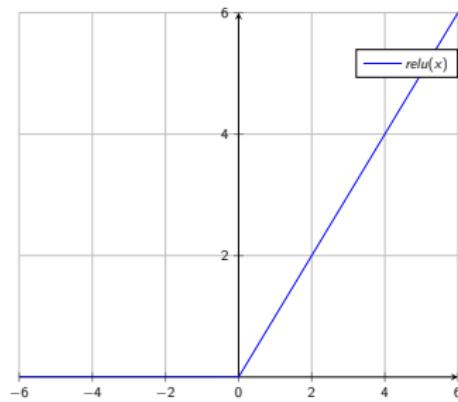
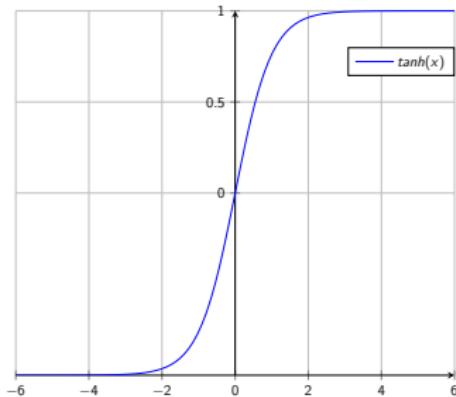
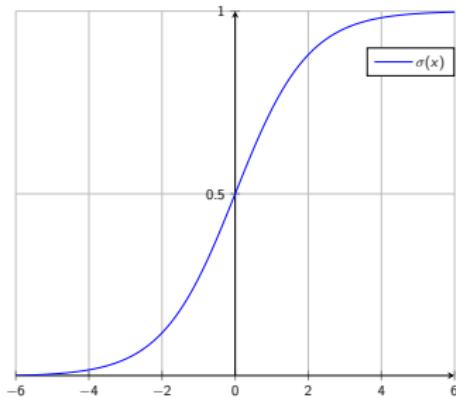
$$g(z) = \tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$

- **Rectified Linear Unit (ReLU)**:

$$g(z) = \max(0, z)$$

- And many more...

# Activation Functions



# Training: Stochastic Gradient Descent (SGD) recap

**Input:**  $D_{train} = \{(x_1, y_1) \dots (x_M, y_M)\}$ ,  $D_{val} = \{(x_1, y_1) \dots (x_D, y_D)\}$ ,  
learning rate  $\eta$ , epochs  $e$ , tolerance  $t$

initialize  $\mathbf{w}$  with zeros

**for each** epoch  $e$  **do**

randomise order in  $D_{train}$

**for each**  $(x_i, y_i)$  in  $D_{train}$  **do**

update  $\mathbf{w} = \mathbf{w} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}; x_i; y_i)$

monitor training and validation loss

**if** previous validation loss – current validation loss; smaller than  $t$

**break**

**return**  $\mathbf{w}$

# Training: Stochastic Gradient Descent (SGD) recap

**Input:**  $D_{train} = \{(x_1, y_1) \dots (x_M, y_M)\}$ ,  $D_{val} = \{(x_1, y_1) \dots (x_D, y_D)\}$ ,  
learning rate  $\eta$ , epochs  $e$ , tolerance  $t$

initialize  $\mathbf{w}$  with zeros

**for each** epoch  $e$  **do**

randomise order in  $D_{train}$

**for each**  $(x_i, y_i)$  in  $D_{train}$  **do**

update  $\mathbf{w} = \mathbf{w} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}; x_i; y_i)$

monitor training and validation loss

**if** previous validation loss – current validation loss; smaller than  $t$

**break**

**return**  $\mathbf{w}$

How to compute the gradient for the weights of the hidden layers?

## Training: Backpropagation Algorithm

- **Forward Pass:** Compute and store all the output values of all the hidden units (for each hidden layer) and the output layer

## Training: Backpropagation Algorithm

- **Forward Pass:** Compute and store all the output values of all the hidden units (for each hidden layer) and the output layer
- **Backward Pass:** Compute the gradients for the output and hidden layers with respect to the cost function  $L$  and update the weights for each layer

# Training: SGD and Backpropagation

**Input:**  $D_{train} = \{(x_1, y_1) \dots (x_M, y_M)\}$ ,  $D_{val} = \{(x_1, y_1) \dots (x_D, y_D)\}$ ,  
learning rate  $\eta$ , epochs  $e$ , tolerance  $t$   
*initialise  $W_i \in W = \{W_1, \dots, W_l\}$  for each layer (small random values)*  
**for each epoch e do**  
    *randomise order in  $D_{train}$*   
    **for each**  $(x_i, y_i)$  in  $D_{train}$  **do**  
        *layer\_outputs = forward\_pass( $(x_i, y_i)$ ,  $W$ )*  
         *$W = backward\_pass((x_i, y_i), W, L, layer\_outputs)$*   
    *monitor training and validation loss*  
    **if** prev val loss – current val loss; smaller than  $t$  : **break**  
**return**  $W$

## Forward Pass

Propagate the inputs forward to compute the outputs for all layers:

$$\mathbf{h}_0 \leftarrow \mathbf{x} \text{ (input layer)}$$

## Forward Pass

Propagate the inputs forward to compute the outputs for all layers:

```
h0 ← x (input layer)  
for layer k = 1, ..., l do
```

## Forward Pass

Propagate the inputs forward to compute the outputs for all layers:

```
h0 ← x (input layer)
for layer  $k = 1, \dots, l$  do
    z $k$  ← W $k$ h $k-1$ 
    h $k$  ←  $g(z)$ 
end for
```

## Forward Pass

Propagate the inputs forward to compute the outputs for all layers:

```
h0 ← x (input layer)
for layer  $k = 1, \dots, l$  do
    z $k$  ← W $k$ h $k-1$ 
    h $k$  ←  $g(\mathbf{z})$ 
end for
Get prediction  $\hat{\mathbf{y}} = \mathbf{h}_l$ 
Compute cross-entropy loss  $L(\hat{\mathbf{y}}, \mathbf{y})$ 
return h, z for all layers
```

## Backward Pass

Propagate the gradients backwards from the loss to the input layer  
(i.e. how each layer's output should change to reduce error):

Compute gradient on the output layer  $\mathbf{g} \leftarrow \nabla_{\hat{y}} L$

## Backward Pass

Propagate the gradients backwards from the loss to the input layer  
(i.e. how each layer's output should change to reduce error):

Compute gradient on the output layer  $\mathbf{g} \leftarrow \nabla_{\hat{y}} L$

**for** layer  $k = l, l - 1, \dots, 1$  **do**

Convert the gradient on the layer's output ( $\mathbf{h}$ ) into a gradient  
before the activation function ( $\mathbf{z}$ ):

$\mathbf{g} \leftarrow \nabla_{\mathbf{z}_k} L = \mathbf{g} \odot f'(\mathbf{z}_k)$  ( $\odot$  element-wise,  $f'(\cdot)$  deriv.)

## Backward Pass

Propagate the gradients backwards from the loss to the input layer  
(i.e. how each layer's output should change to reduce error):

Compute gradient on the output layer  $\mathbf{g} \leftarrow \nabla_{\hat{y}} L$

**for** layer  $k = l, l - 1, \dots, 1$  **do**

    Convert the gradient on the layer's output ( $\mathbf{h}$ ) into a gradient  
    before the activation function ( $\mathbf{z}$ ):

$\mathbf{g} \leftarrow \nabla_{z_k} L = \mathbf{g} \odot f'(\mathbf{z}_k)$  ( $\odot$  element-wise,  $f'(\cdot)$  deriv.)

    Compute gradients on weights:

$\nabla_{W_k} L = \mathbf{g} h_{k-1}$

## Backward Pass

Propagate the gradients backwards from the loss to the input layer  
(i.e. how each layer's output should change to reduce error):

Compute gradient on the output layer  $\mathbf{g} \leftarrow \nabla_{\hat{y}} L$

**for** layer  $k = l, l - 1, \dots, 1$  **do**

Convert the gradient on the layer's output ( $\mathbf{h}$ ) into a gradient  
before the activation function ( $\mathbf{z}$ ):

$\mathbf{g} \leftarrow \nabla_{\mathbf{z}_k} L = \mathbf{g} \odot f'(\mathbf{z}_k)$  ( $\odot$  element-wise,  $f'(\cdot)$  deriv.)

Compute gradients on weights:

$\nabla_{W_k} L = \mathbf{g} h_{k-1}$

Compute the gradients w.r.t. the next hidden layer:

$\mathbf{g} \leftarrow \nabla_{h_{k-1}} L = \mathbf{g} W_k$

## Backward Pass

Propagate the gradients backwards from the loss to the input layer  
(i.e. how each layer's output should change to reduce error):

Compute gradient on the output layer  $\mathbf{g} \leftarrow \nabla_{\hat{y}} L$

**for** layer  $k = l, l - 1, \dots, 1$  **do**

    Convert the gradient on the layer's output ( $\mathbf{h}$ ) into a gradient  
    before the activation function ( $\mathbf{z}$ ):

$\mathbf{g} \leftarrow \nabla_{\mathbf{z}_k} L = \mathbf{g} \odot f'(\mathbf{z}_k)$  ( $\odot$  element-wise,  $f'(\cdot)$  deriv.)

    Compute gradients on weights:

$\nabla_{W_k} L = \mathbf{g} h_{k-1}$

    Compute the gradients w.r.t. the next hidden layer:

$\mathbf{g} \leftarrow \nabla_{h_{k-1}} L = \mathbf{g} W_k$

    Update current weights:

$W_k \leftarrow W_k - \eta \nabla_{W_k} L$

**end for**

**return**  $W$

# Regularisation

- **L2-regularisation** in the weights of each layer (added in the loss function of each layer)

# Regularisation

- **L2-regularisation** in the weights of each layer (added in the loss function of each layer)
- **Dropout**: randomly ignore a percentage (e.g. 20% or 50%) of layer outputs during training:

# Regularisation

- **L2-regularisation** in the weights of each layer (added in the loss function of each layer)
- **Dropout:** randomly ignore a percentage (e.g. 20% or 50%) of layer outputs during training:
  - Apply a random binary mask after the activation function, i.e. elementwise multiplication with vector containing %20 0s in random positions

# Design Choices

- How many layers?
- How many units per layer?
- What activation function(s)?

# Design Choices

- How many layers?
- How many units per layer?
- What activation function(s)?
- Architecture engineering vs feature engineering
- Theory says that we can approximate any function with one hidden layer, practice says different architectures work well for different problems

# Implementation tips

- Learning objective non-convex: initialisation matters
  - start with small non-zero values
  - random restarts to escape local optima
- Greater learning capacity makes overfitting more likely: regularise
- Many open libraries are available: PyTorch, Tensorflow, MxNet, Keras etc.

## Applications: Word Vectors

- Lecture 1: word vectors by **counting** co-occurrences with context words

## Applications: Word Vectors

- Lecture 1: word vectors by **counting** co-occurrences with context words
- Instead, use a feedforward network to **predict** a context word for a given word (and vice versa)

## Applications: Word Vectors

- Lecture 1: word vectors by **counting** co-occurrences with context words
- Instead, use a feedforward network to **predict** a context word for a given word (and vice versa)
- **Word2Vec (Mikolov et al., 2013)** family, more recently supporting char n-grams (e.g. FastText)

# Word2Vec

- **Skip-gram model:** Given a word predict its context words

# Word2Vec

- **Skip-gram model:** Given a word predict its context words
- **Continuous BOW (CBOW):** Given context words predict the current word

# Word2Vec

- **Skip-gram model:** Given a word predict its context words
- **Continuous BOW (CBOW):** Given context words predict the current word
- **Input:** A word, represented as a one-hot vector over vocabulary (or the vocabulary index for memory efficiency!)

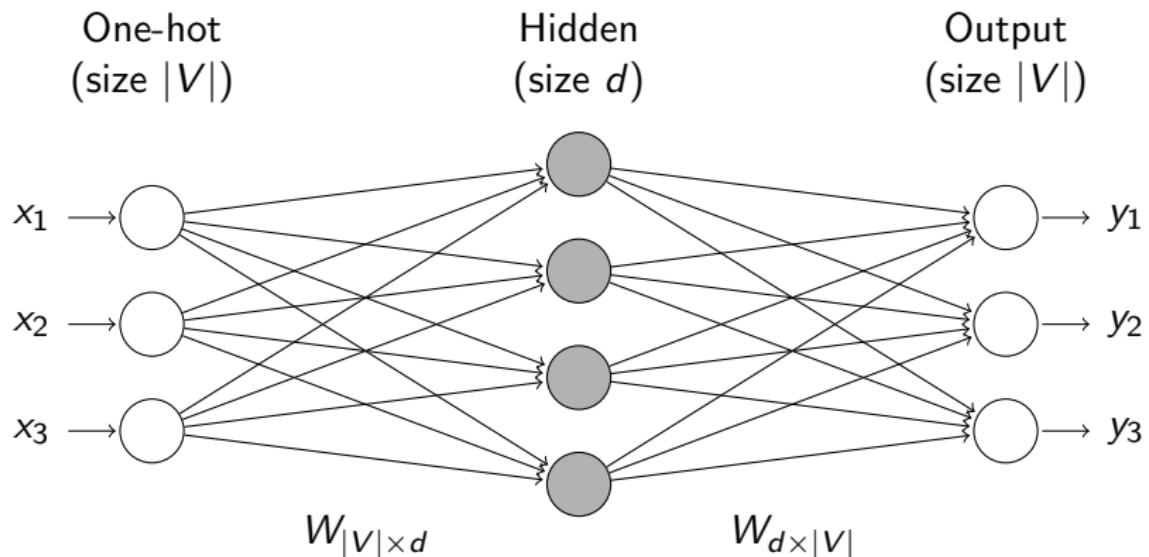
# Word2Vec

- **Skip-gram model:** Given a word predict its context words
- **Continuous BOW (CBOW):** Given context words predict the current word
- **Input:** A word, represented as a one-hot vector over vocabulary (or the vocabulary index for memory efficiency!)
- **Hidden layer:** One hidden layer of size vocabulary  $\times$  hidden size (usually 300), linear activation function

# Word2Vec

- **Skip-gram model:** Given a word predict its context words
- **Continuous BOW (CBOW):** Given context words predict the current word
- **Input:** A word, represented as a one-hot vector over vocabulary (or the vocabulary index for memory efficiency!)
- **Hidden layer:** One hidden layer of size vocabulary  $\times$  hidden size (usually 300), linear activation function
- **Output:** softmax over the vocabulary to predict the correct context/target words respectively

# Word2Vec Architecture



# Word2Vec

- Training data  $(x, y)$  can be obtained from large corpora

# Word2Vec

- Training data  $(x, y)$  can be obtained from large corpora
- For Skip-gram:

the **cat** sat  $\rightarrow$  cat, {the, sat}

# Word2Vec

- Training data ( $x, y$ ) can be obtained from large corpora
- For Skip-gram:

the **cat** sat  $\rightarrow$  cat, {the, sat}

- For CBOW:

the **cat** sat  $\rightarrow$  {the, sat}, cat

# Word2Vec

- Training data  $(x, y)$  can be obtained from large corpora
- For Skip-gram:

the **cat** sat  $\rightarrow$  cat, {the, sat}

- For CBOW:

the **cat** sat  $\rightarrow$  {the, sat}, cat

- **Vector** of a word  $x_i = W_i$ , from the network weights

# Word2Vec

- Training data  $(x, y)$  can be obtained from large corpora
- For Skip-gram:

the **cat** sat  $\rightarrow$  cat, {the, sat}

- For CBOW:

the **cat** sat  $\rightarrow$  {the, sat}, cat

- **Vector** of a word  $x_i = W_i$ , from the network weights
- Evaluation: standard approaches for word representation (see Lecture 1)

# Word2Vec

- Training data ( $x, y$ ) can be obtained from large corpora
- For Skip-gram:

the **cat** sat → cat, {the, sat}

- For CBOW:

the **cat** sat → {the, sat}, cat

- **Vector** of a word  $x_i = W_i$ , from the network weights
- Evaluation: standard approaches for word representation (see Lecture 1)
- Pre-trained word embeddings are widely re-used in other NLP tasks, i.e. transfer learning (more in Lecture 10)

## Word2Vec: Implementation Details

- Word2Vec is a huge neural network, to make the training feasible:

# Word2Vec: Implementation Details

- Word2Vec is a huge neural network, to make the training feasible:
  - **Negative Sampling:** Update the weights for the positive word, plus the weights for a small number (5-20) other words that we want to output 0

# Word2Vec: Implementation Details

- Word2Vec is a huge neural network, to make the training feasible:
  - **Negative Sampling:** Update the weights for the positive word, plus the weights for a small number (5-20) other words that we want to output 0
  - **Subsampling frequent words** to decrease the number of training examples

# Applications: Text Classification

- **Approach 1:** Pass BOW vectors into a series of hidden layers  
(extending the LR model in Lecture 2)

## Applications: Text Classification

- **Approach 1:** Pass BOW vectors into a series of hidden layers (extending the LR model in Lecture 2)
- **Approach 2:** Pass one-hot word vectors through an **embedding layer** to obtain embeddings for each word in a document which are subsequently **concatenated (or added/averaged) and passed through a series of hidden layers**

## Applications: Text Classification

- **Approach 1:** Pass BOW vectors into a series of hidden layers (extending the LR model in Lecture 2)
- **Approach 2:** Pass one-hot word vectors through an **embedding layer** to obtain embeddings for each word in a document which are subsequently **concatenated (or added/averaged) and passed through a series of hidden layers**
- Approach 2 is more contemporary and usually the embedding layer is pre-trained (e.g. using Word2Vec) and is not updated during training

# Bibliography

- Chapters 6-8 from Goodfellow et al.
- Sections 3-6 from Goldberg
- Tutorial on backprop by D. Stansbury
- Word2vec tutorial by Chris McCormick

Coming up next..

- Recurrent Networks and Neural Language Modelling

# Recurrent Neural Networks and Neural LMs

## COM6513 Natural Language Processing

Nikos Aletras

[n.aletras@sheffield.ac.uk](mailto:n.aletras@sheffield.ac.uk)

Computer Science Department

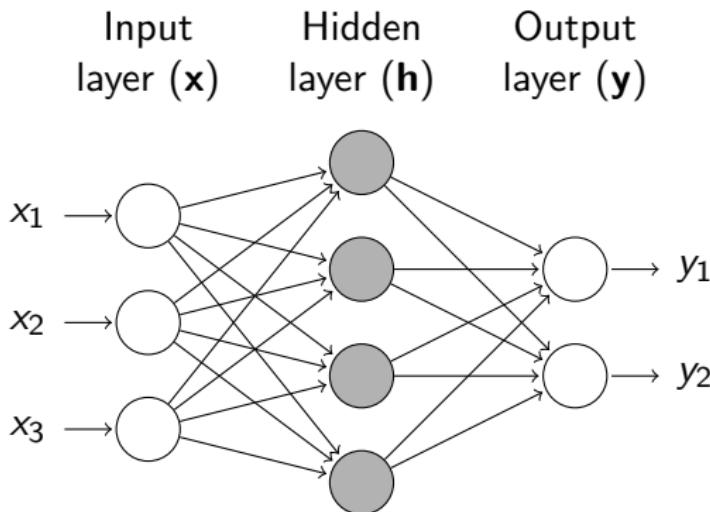
Week 7  
Spring 2022



## In lecture 6...

- **Feedforward Neural Networks** and how to train them with **Backprop**
- Feedforward nets are useful to learn word representations but they ignore word order and dependencies between words in a given document.

# Feedforward Neural Network



$$\begin{aligned}\mathbf{h} &= g(\mathbf{W}_h \mathbf{x}) \\ \mathbf{y} &= \text{softmax}(\mathbf{W}_o \mathbf{h}) \\ \mathbf{W}_o &\in \mathcal{R}^{h \times y}\end{aligned}$$

$g(\cdot)$  is an activation function

In this lecture...

- **Recurrent Neural Networks (RNNs)** to capture long-range dependencies in a document

In this lecture...

- **Recurrent Neural Networks (RNNs)** to capture long-range dependencies in a document
- Train with **SGD** and **Backpropagation through Time**

## In this lecture...

- **Recurrent Neural Networks (RNNs)** to capture long-range dependencies in a document
- Train with **SGD** and **Backpropagation through Time**
- RNN extensions: **Long-Short Term Memory (LSTM)** and **Gated-Recurrent Unit (GRU)**

## In this lecture...

- **Recurrent Neural Networks (RNNs)** to capture long-range dependencies in a document
- Train with **SGD** and **Backpropagation through Time**
- RNN extensions: **Long-Short Term Memory (LSTM)** and **Gated-Recurrent Unit (GRU)**
- Language modelling: return sentence probabilities as well as representations

## In this lecture...

- **Recurrent Neural Networks (RNNs)** to capture long-range dependencies in a document
- Train with **SGD** and **Backpropagation through Time**
- RNN extensions: **Long-Short Term Memory (LSTM)** and **Gated-Recurrent Unit (GRU)**
- Language modelling: return sentence probabilities as well as representations
- Text classification: learn contextualised word representations and use them to predict a given class

## In this lecture...

- **Recurrent Neural Networks (RNNs)** to capture long-range dependencies in a document
- Train with **SGD** and **Backpropagation through Time**
- RNN extensions: **Long-Short Term Memory (LSTM)** and **Gated-Recurrent Unit (GRU)**
- Language modelling: return sentence probabilities as well as representations
- Text classification: learn contextualised word representations and use them to predict a given class
- Improve RNNs with **Attention**

# Neural Language Modelling: Problem Setup

Training data is a (large) set of word sequences:

$$D_{train} = \{\mathbf{x}^1, \dots, \mathbf{x}^M\}$$
$$\mathbf{x} = [x_1, \dots, x_N]$$

# Neural Language Modelling: Problem Setup

Training data is a (large) set of word sequences:

$$D_{train} = \{\mathbf{x}^1, \dots, \mathbf{x}^M\}$$
$$\mathbf{x} = [x_1, \dots, x_N]$$

for example:

$$\mathbf{x} = [its, water, is, so, transparent, STOP]$$

# Neural Language Modelling: Problem Setup

Training data is a (large) set of word sequences:

$$D_{train} = \{\mathbf{x}^1, \dots, \mathbf{x}^M\}$$
$$\mathbf{x} = [x_1, \dots, x_N]$$

for example:

$$\mathbf{x} = [its, water, is, so, transparent, STOP]$$

We want to learn a model that returns:

$$P(\mathbf{x}), \text{ for } \forall \mathbf{x} \in V^{maxN}$$

$V$  is the vocabulary and  $V^{maxN}$  all possible sentences

# Language modelling as classification

$$\begin{aligned} P(\mathbf{x}) &= P(x_1, \dots, x_N) \\ &= P(x_1)P(x_2 \dots x_N | x_1) \\ &= P(x_1)P(x_2 | x_1) \dots P(x_N | x_1, \dots, x_{N-1}) \end{aligned}$$

# Language modelling as classification

$$\begin{aligned}P(\mathbf{x}) &= P(x_1, \dots, x_N) \\&= P(x_1)P(x_2 \dots x_N | x_1) \\&= P(x_1)P(x_2 | x_1) \dots P(x_N | x_1, \dots, x_{N-1})\end{aligned}$$

Let's write the probabilities as LR (remember the CRF?):

$$p(x_n = k | x_{n-1} \dots x_1) = \frac{\exp(\mathbf{w}_k \cdot \phi(x_{n-1} \dots x_1))}{\sum_{k'=1}^{|\mathcal{V}|} \exp(\mathbf{w}_{k'} \cdot \phi(x_{n-1} \dots x_1))}$$

- $\mathbf{w}_k$  are the weights for word  $k$
- $\phi(x_{n-1} \dots x_1)$  are the features extracted from the previous words (one-hot encoding of  $x_{n-1} \dots x_1$ )

# Representing word sequences

Looks like a neural network:

$$p(x_n | x_{n-1} \dots x_1) = \text{softmax}(\mathbf{W}\phi(x_{n-1} \dots x_1))$$

$\mathbf{W} \in \mathcal{R}^{|\mathcal{V}| \times |\mathcal{C}|}$  has weights for each word and context

# Representing word sequences

Looks like a neural network:

$$p(x_n | x_{n-1} \dots x_1) = \text{softmax}(\mathbf{W}\phi(x_{n-1} \dots x_1))$$

$\mathbf{W} \in \mathcal{R}^{|\mathcal{V}| \times |\mathcal{C}|}$  has weights for each word and context

Let's represent the context with a vector  $s_{n-1} \in \mathcal{R}^d$ :

$$p(x_n | x_{n-1} \dots x_1) = \text{softmax}(\mathbf{V}s_{n-1})$$

# Representing word sequences

Looks like a neural network:

$$p(x_n | x_{n-1} \dots x_1) = \text{softmax}(\mathbf{W}\phi(x_{n-1} \dots x_1))$$

$\mathbf{W} \in \mathcal{R}^{|\mathcal{V}| \times |\mathcal{C}|}$  has weights for each word and context

Let's represent the context with a vector  $s_{n-1} \in \mathcal{R}^d$ :

$$p(x_n | x_{n-1} \dots x_1) = \text{softmax}(\mathbf{V}s_{n-1})$$

$\mathbf{V} \in \mathcal{R}^{|\mathcal{V}| \times d}$  maps the context to a probability distribution over the words.

# Representing word sequences

Looks like a neural network:

$$p(x_n | x_{n-1} \dots x_1) = \text{softmax}(\mathbf{W}\phi(x_{n-1} \dots x_1))$$

$\mathbf{W} \in \mathcal{R}^{|\mathcal{V}| \times |\mathcal{C}|}$  has weights for each word and context

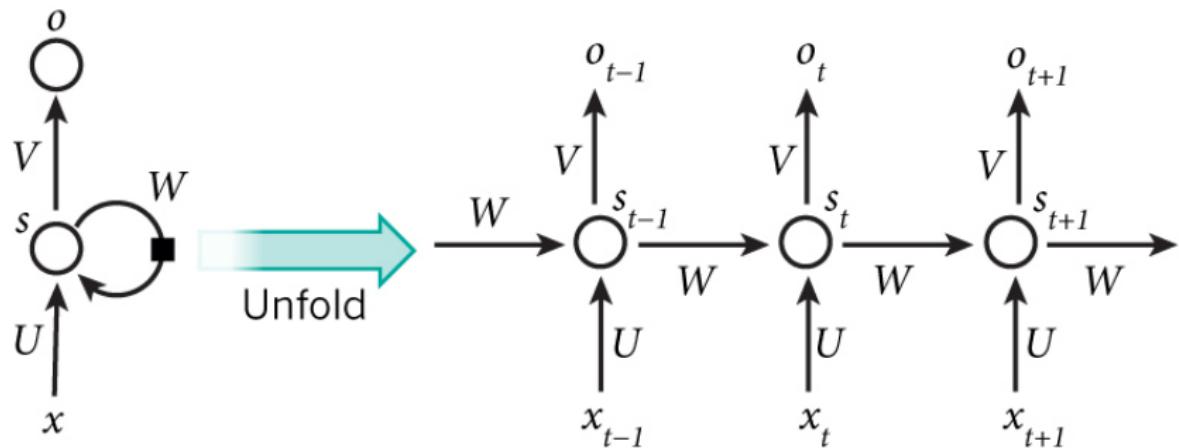
Let's represent the context with a vector  $s_{n-1} \in \mathcal{R}^d$ :

$$p(x_n | x_{n-1} \dots x_1) = \text{softmax}(\mathbf{V}s_{n-1})$$

$\mathbf{V} \in \mathcal{R}^{|\mathcal{V}| \times d}$  maps the context to a probability distribution over the words.

How do we get  $s_{n-1}$ ?

# Recurrent neural networks



When generating,  $x_t$  is the highest-scoring word in  $o_{t-1}$

# Recurrent Neural Networks

$$s_n = \sigma(\mathbf{W}s_{n-1} + \mathbf{U}x_n)$$

- $s_{n-1} \in \mathcal{R}^d$ : "memory" of the context until word  $x_{n-1}$
- $\mathbf{W} \in \mathcal{R}^{d \times d}$ : controls how this memory is passed on
- $\mathbf{U} \in \mathcal{R}^{|\mathcal{V}| \times d}$ : matrix containing the word vectors for all the words,  $x_n$  picks one

# Recurrent Neural Networks

$$s_n = \sigma(\mathbf{W}s_{n-1} + \mathbf{U}x_n)$$

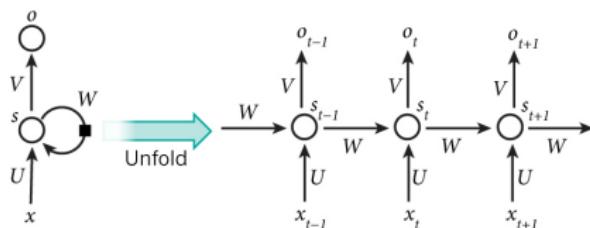
- $s_{n-1} \in \mathcal{R}^d$ : "memory" of the context until word  $x_{n-1}$
- $\mathbf{W} \in \mathcal{R}^{d \times d}$ : controls how this memory is passed on
- $\mathbf{U} \in \mathcal{R}^{|\mathcal{V}| \times d}$ : matrix containing the word vectors for all the words,  $x_n$  picks one

To get the probability distribution for word  $x_n$ :

$$\mathbf{o}_{n-1} = p(x_n | x_{n-1} \dots x_1) = \text{softmax}(\mathbf{V}s_{n-1})$$

- $\mathbf{V} \in \mathcal{R}^{d \times |\mathcal{V}|}$ : output weight matrix

# Training RNNs



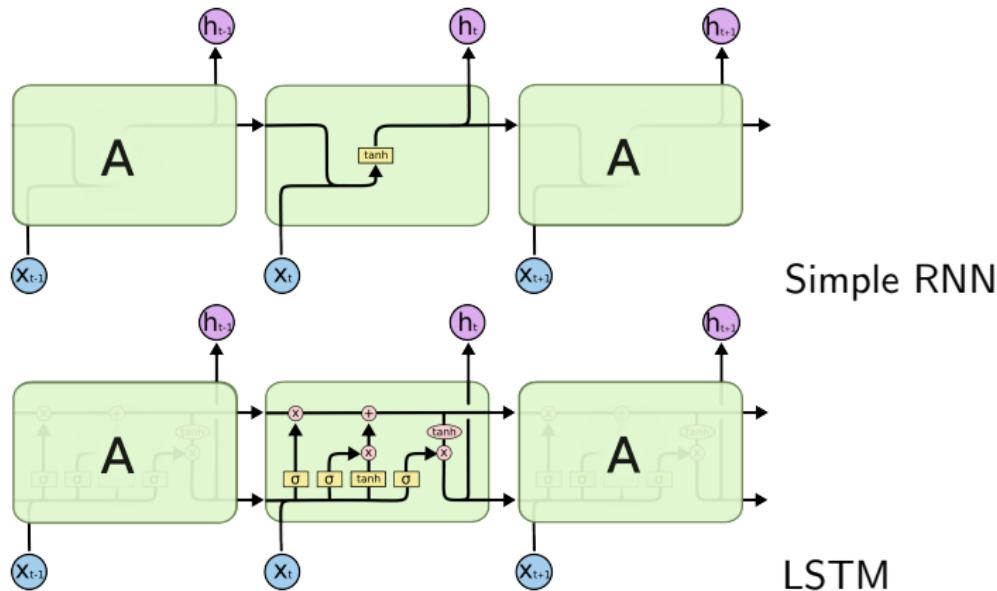
- We need to learn the word vectors  $\mathbf{U}$ , hidden and output layer parameters  $\mathbf{W}, \mathbf{V}$
- Standard backpropagation can't work because of the recurrence: we reuse the hidden layer parameters  $\mathbf{W}$
- **Backpropagation Through Time:** unroll the graph for  $n$  steps and sum the gradients in updating
- Not as restrictive as the  $n$ th-order Markov: we still use all previous words through the recurrence.

# Limitations of RNNs

RNNs can't capture long-range dependencies:

- effectively have one layer per word in the sentence
- all context information has to be passed by the hidden layer
- vanishing gradients: the gradient from the last word often never reaches the first

# Long-Short Term Memory (LSTM) network<sup>1</sup>



A memory cell is used in addition to the hidden layer to control what information from previous timesteps is useful in predicting.

<sup>1</sup>Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

# Long-Short Term Memory (LSTM) network

- **Forget gate** (what info to throw away from previous steps):

$$f_t = \sigma(W_f[h_{t-1}, x_t])$$

# Long-Short Term Memory (LSTM) network

- **Forget gate** (what info to throw away from previous steps):

$$f_t = \sigma(W_f[h_{t-1}, x_t])$$

- **Input gate** (what new info will be stored in the memory cell):

$$i_t = \sigma(W_i[h_{t-1}, x_t])$$

# Long-Short Term Memory (LSTM) network

- **Forget gate** (what info to throw away from previous steps):

$$f_t = \sigma(W_f[h_{t-1}, x_t])$$

- **Input gate** (what new info will be stored in the memory cell):

$$i_t = \sigma(W_i[h_{t-1}, x_t])$$

- New **memory cell candidate values**:

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t])$$

# Long-Short Term Memory (LSTM) network

- **Forget gate** (what info to throw away from previous steps):

$$f_t = \sigma(W_f[h_{t-1}, x_t])$$

- **Input gate** (what new info will be stored in the memory cell):

$$i_t = \sigma(W_i[h_{t-1}, x_t])$$

- New **memory cell candidate values**:

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t])$$

- Update memory cell (using input and output gates):

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

# Long-Short Term Memory (LSTM) network

- **Output** (decide what's the output filtered by the memory cell):

$$o_t = \sigma(W_o[h_{t-1}, x_t])$$

$$h_t = o_t * \tanh(C_t)$$

# Gated Recurrent Unit (GRU<sup>2</sup>)

- LSTM variant

---

<sup>2</sup>Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

# Gated Recurrent Unit (GRU<sup>2</sup>)

- LSTM variant
- **Update gate** (combines input and forget gates):

$$z_t = \sigma(W_z[h_{t-1}, x_t])$$

---

<sup>2</sup>Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

# Gated Recurrent Unit (GRU<sup>2</sup>)

- LSTM variant
- **Update gate** (combines input and forget gates):

$$z_t = \sigma(W_z[h_{t-1}, x_t])$$

- **Recurrent state** (merges cell state with hidden state):

$$r_t = \sigma(W_r[h_{t-1}, x_t])$$

---

<sup>2</sup>Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

# Gated Recurrent Unit (GRU<sup>2</sup>)

- LSTM variant
- **Update gate** (combines input and forget gates):

$$z_t = \sigma(W_z[h_{t-1}, x_t])$$

- **Recurrent state** (merges cell state with hidden state):

$$r_t = \sigma(W_r[h_{t-1}, x_t])$$

- New **output candidate values**:

$$\tilde{h}_t = \tanh(W[r_t * h_{t-1}, x_t])$$

---

<sup>2</sup>Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

# Gated Recurrent Unit (GRU<sup>2</sup>)

- LSTM variant
- **Update gate** (combines input and forget gates):

$$z_t = \sigma(W_z[h_{t-1}, x_t])$$

- **Recurrent state** (merges cell state with hidden state):

$$r_t = \sigma(W_r[h_{t-1}, x_t])$$

- New **output candidate values**:

$$\tilde{h}_t = \tanh(W[r_t * h_{t-1}, x_t])$$

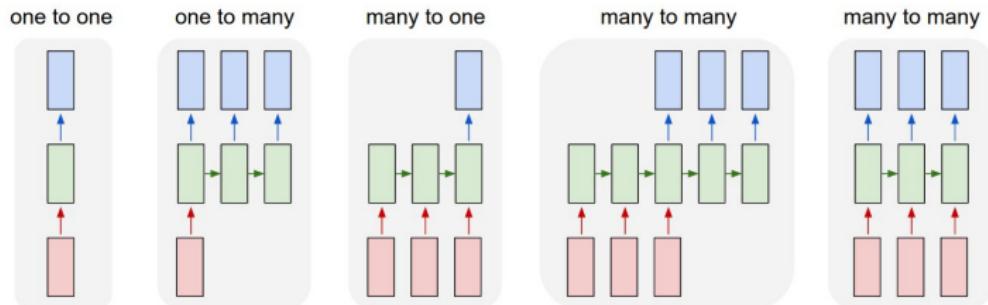
- Output:

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

---

<sup>2</sup>Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

# RNN Architecture Variants in NLP



<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

- many to one: e.g. text classification
- many to many (equal): e.g. PoS tagging
- many to many (unequal): e.g. machine translation (coming next week), language generation, summarisation

# Representation learning with RNNs

- RNNs learn word and **sentence/document representations**
- Words are not as interesting since RNNs are slower to train than Skip-Gram: thus use less data
- hint: use pre-trained word vectors (e.g. skipgram) to initialise the RNN word vectors
- RNN sentence/document representations though are used often!
- Bi-directional RNNs can also be used to learn document representations: one RNN parsing the input from start to end and another one from end to start.

# Improving RNNs with Attention

- In many-to-one tasks (e.g. text classification), usually the outputs from each timestep are combined (concatenated/averaged/summed) and then passed to the output layer.

# Improving RNNs with Attention

- In many-to-one tasks (e.g. text classification), usually the outputs from each timestep are combined (concatenated/averaged/summed) and then passed to the output layer.
- This **naive combination** assumes that all representations (from each timestep) have **equal contribution**.

# Improving RNNs with Attention

- In many-to-one tasks (e.g. text classification), usually the outputs from each timestep are combined (concatenated/averaged/summed) and then passed to the output layer.
- This **naive combination** assumes that all representations (from each timestep) have **equal contribution**.
- But that might not be the case!

# Improving RNNs with Attention

- In many-to-one tasks (e.g. text classification), usually the outputs from each timestep are combined (concatenated/averaged/summed) and then passed to the output layer.
- This **naive combination** assumes that all representations (from each timestep) have **equal contribution**.
- But that might not be the case!
- **Attention:** compute a weighted linear combination of all the contextualised representations obtained from the RNN:

$$\mathbf{c} = \sum_i \mathbf{h}_i \alpha_i$$

# Improving RNNs with Attention

- In many-to-one tasks (e.g. text classification), usually the outputs from each timestep are combined (concatenated/averaged/summed) and then passed to the output layer.
- This **naive combination** assumes that all representations (from each timestep) have **equal contribution**.
- But that might not be the case!
- **Attention:** compute a weighted linear combination of all the contextualised representations obtained from the RNN:

$$\mathbf{c} = \sum_i \mathbf{h}_i \alpha_i$$

- Pass  $\mathbf{c}$  to the output layer for classification

# Attention Mechanism

- Attention usually consists of a similarity function  $\phi$  followed by softmax:

$$a_i = \frac{\exp(\phi(\mathbf{h}_i, \mathbf{q}))}{\sum_{k=1}^t \exp(\phi(\mathbf{q}, \mathbf{h}_k))}$$

---

<sup>3</sup>Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

<sup>4</sup>Vaswani, Ashish et al. (2017). Attention is all you need. In Advances in neural information processing systems, pp. 5998-6008.

# Attention Mechanism

- Attention usually consists of a similarity function  $\phi$  followed by softmax:

$$a_i = \frac{\exp(\phi(\mathbf{h}_i, \mathbf{q}))}{\sum_{k=1}^t \exp(\phi(\mathbf{q}, \mathbf{h}_k))}$$

- $\mathbf{q} \in R^N$  is a trainable vector (learns task specific information)

---

<sup>3</sup>Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

<sup>4</sup>Vaswani, Ashish et al. (2017). Attention is all you need. In Advances in neural information processing systems, pp. 5998-6008.

# Attention Mechanism

- Attention usually consists of a similarity function  $\phi$  followed by softmax:

$$a_i = \frac{\exp(\phi(\mathbf{h}_i, \mathbf{q}))}{\sum_{k=1}^t \exp(\phi(\mathbf{q}, \mathbf{h}_k))}$$

- $\mathbf{q} \in R^N$  is a trainable vector (learns task specific information)
- Additive (or tanh):<sup>3</sup>

$$\phi(h_i, \mathbf{q}) = \mathbf{q}^T \tanh(W\mathbf{h}_i)$$

---

<sup>3</sup>Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

<sup>4</sup>Vaswani, Ashish et al. (2017). Attention is all you need. In Advances in neural information processing systems, pp. 5998-6008.

# Attention Mechanism

- Attention usually consists of a similarity function  $\phi$  followed by softmax:

$$a_i = \frac{\exp(\phi(\mathbf{h}_i, \mathbf{q}))}{\sum_{k=1}^t \exp(\phi(\mathbf{q}, \mathbf{h}_k))}$$

- $\mathbf{q} \in R^N$  is a trainable vector (learns task specific information)
- Additive (or tanh):<sup>3</sup>

$$\phi(\mathbf{h}_i, \mathbf{q}) = \mathbf{q}^T \tanh(W\mathbf{h}_i)$$

- Scaled Dot-Product:<sup>4</sup>

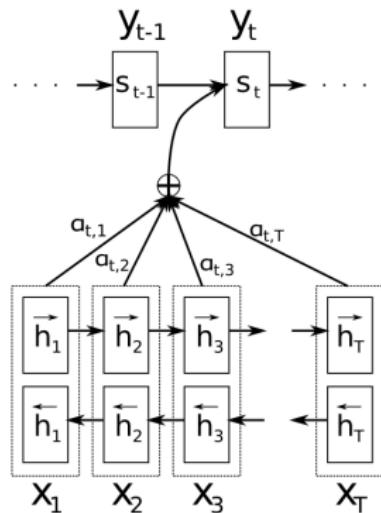
$$\phi(\mathbf{h}_i, \mathbf{q}) = \frac{\mathbf{h}_i^T \mathbf{q}}{\sqrt{N}}$$

---

<sup>3</sup>Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

<sup>4</sup>Vaswani, Ashish et al. (2017). Attention is all you need. In Advances in neural information processing systems, pp. 5998-6008.

# Attention Mechanism



<http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/>

# Bibliography

- Chapters 6-8 from Goodfellow et al.
- Section 6.3 from Eisenstein
- Section 10 from Goldberg
- Blog post on RNNs
- Blog post on LSTMs from where some of the figures were taken
- Blog post on attention

## Coming up next..

- Sequence-to-Sequence models and Machine Translation by Dr. Fernando Alva
- Information Extraction and Ethics in NLP by Prof. Jochen Leidner

# Information Extraction

## Natural Language Processing Module 2022

Dr Samuel Mensah  
[s.mensah@sheffield.ac.uk](mailto:s.mensah@sheffield.ac.uk)



The  
University  
Of  
Sheffield.

March 29, 2022

Copyright ©2019 by Prof Jochen L. Leidner & Dr Nikos Aletras.  
All rights reserved.

# Outline

- Information Extraction Background
- System Architecture
- Feature Extraction
- Feature Learning
- Rule- and Machine-learning based methods
- Experimental Methodology
- Information Extraction Tasks:
  - Named Entity Recognition
  - Relation Extraction
- Application: Information Extraction for Question Answering

In this session, we aim to:

- get introduced to information extraction (IE), its concepts and history
- learn about rule based and learning approaches to IE
- application of IE in question answering

## **Question:**

When was the Swedish orchestra Kungliga Hovkapellet established?

# Motivation

## Question:

When was the Swedish orchestra Kungliga Hovkapellet established?



When was the Swedish orchestra Kungliga Hovkapellet established?

All

News

Images

Maps

Videos

More

Tools

About 8,740 results (0.54 seconds)

**1526**

Kungliga Hovkapellet (The Royal Swedish Orchestra) has been at the heart of Swedish culture for almost 500 years. Founded in **1526** as the orchestra to the Royal Court, it became the orchestra of the Royal Swedish Opera in 1773.

# “Information Extraction” Defined

What is Information Extraction (IE)?

# "Information Extraction" Defined

## What is Information Extraction (IE)?

- the extraction of structured information from unstructured (= textual) sources.
- a practically-motivated engineering discipline (models not necessarily inspired by nature)
- its significance is connected to the growing amount of information in text and its potential use in systems (e.g. question answering)
- the use of natural language processing techniques to extract relationships between entities in text for knowledge base population; extract events.

# Applications of Information Extraction

## Why Information Extraction?

There are several application domains for IE systems:

- Bio-medical IE applications (genes and proteins)
- Financial IE applications (e.g., Message Formatting Expert (MFE) system in DBS bank, Singapore to extract relevant information from “letters of credit”.)
- Intelligence & IE applications (terrorists & crimes)
- e-Commerce IE applications (brands & productions, sentiment information extraction for recommender systems)
- **Question Answering** (e.g. Google Search)

# A Short History of Information Extraction

- **1965:** NYU "Linguistic String" project (N. Sager) to facilitate search and retrieval of requested information in scientific and technical literature.
- **1982:** DeJong's FRUMP (Fast Reading Understanding and Memory Program) system: "sketchy scripts" to highlight key information in text.
- **1987/1989 MUC I+II:** Naval operation messages
- **1991/1998 MUC 3-7:** News reports and other domains
- **200-2004:** ACE: Automatic Content Extraction
  - defines research objective in terms of the target objects
  - from text spans to abstract entities
  - extract relations (e.g., capital of), event (e.g., destruction)
  - English, Chinese, Arabic
- **2010s:** First neural approaches to IE

*"Kungliga Hovkapellet is a Swedish orchestra, originally part of the Royal Court in Sweden's capital Stockholm. The orchestra originally consisted of both musicians and singers. It had only male members until 1727, when Sophia Schröder and Judith Fischer were employed as vocalists; in the 1850s, the harpist Marie Pauline Åhman became the first female instrumentalist. From 1731, public concerts were performed at Ridderhuset in Stockholm. Since 1773, when the Royal Swedish Opera was founded by Gustav III of Sweden, the Kungliga Hovkapellet has been part of the opera's company."<sup>1</sup>*

---

<sup>1</sup> DocRED: A Large-Scale Document-Level Relation Extraction Dataset

**[“Kungliga Hovkapellet is a Swedish orchestra, originally part of the Royal Court in Sweden’s capital Stockholm. The orchestra originally consisted of both musicians and singers. It had only male members until 1727], when Sophia Schröder and Judith Fischer were employed as vocalists; in the 1850s, the harpist Marie Pauline Åhman became the first female instrumentalist. From 1731, public concerts were performed at Riddarhuset in Stockholm. Since 1773, when the Royal Swedish Opera was founded by Gustav III of Sweden, the Kungliga Hovkapellet has been part of the opera’s company.”**

**[“Kungliga Hovkapellet is a Swedish orchestra, originally part of the Royal Court in Sweden’s capital Stockholm. The orchestra originally consisted of both musicians and singers. It had only male members until 1727], when Sophia Schröder and Judith Fischer were employed as vocalists; in the 1850s, the harpist Marie Pauline Åhman became the first female instrumentalist. From 1731, public concerts were performed at Riddarhuset in Stockholm. Since 1773, when the Royal Swedish Opera was founded by Gustav III of Sweden, the Kungliga Hovkapellet has been part of the opera’s company.”**

- What is the capital of Sweden?
- Where can we find “The Royal Court Orchestra”?
- What had only male members until 1727?

# IE Example: Tasks

By answering these questions, we have implicitly performed a number of IE tasks:

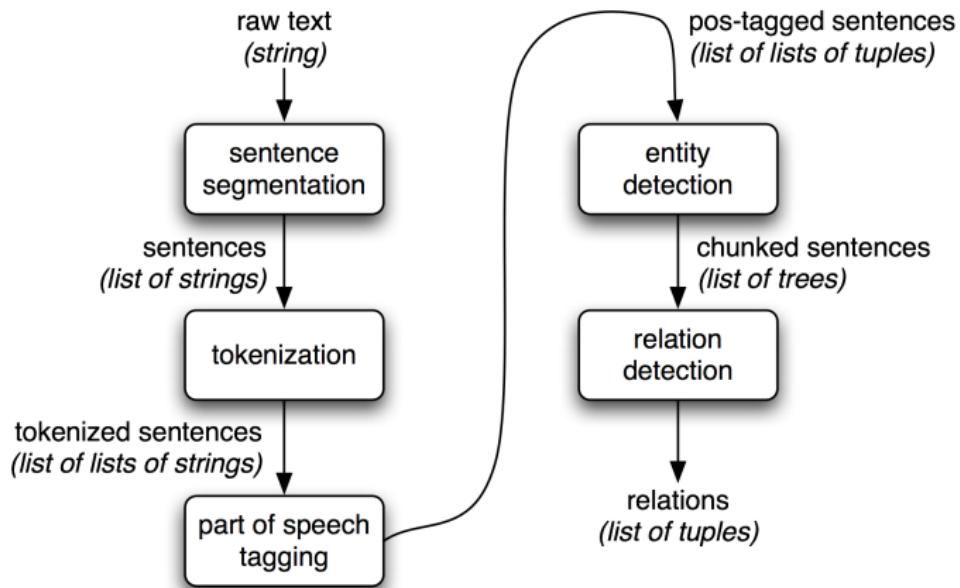
- ① Named Entity Recognition
  - *Stockholm, Sweden, Kungliga Hovkapellet, etc.*
- ② Entity Disambiguation
  - *Kungliga Hovkapellet is also known as "The Royal Court Orchestra"*
- ③ Entity Coherence
  - *The word "It" makes reference to Kungliga Hovkapellet*
- ④ Relation Classification/Extraction
  - *The word "capital" shows the relationship between Sweden and Stockholm*
  - *Structured knowledge: (Stockholm, capital, Sweden)*
- ⑤ Knowledge Base Population

We can automate Question Answering via IE

# Information Extraction System Architecture

- **Text normalization:** Reducing the text into a single canonical form
  - **Tokenization:** Splitting the text into smaller units such as words or characters
  - **Stemming:** Reducing inflectional form of words to their base form (e.g., eating, eats, eaten is reduced to eat)
  - **Lemmatization:** similar to a stemming process but guided by a lexical knowledge base to obtain accurate word stems.
  - **POS tagger:** Assigns part-of-speech tags to words in text
  - **Chunk parser:** individual pieces of text and grouping them into meaningful grammatical chunks or syntactic units.
- **Named entity tagger:** identify and classify entities
- **Relation Tagger:** find relation between entities
- Populate knowledge base with facts

# Information Extraction System Architecture



2

Figure: Pipeline Architecture for an Information Extraction System.

<sup>2</sup><https://www.nltk.org/book/ch07.html>

# Feature Extraction

## Spam Classification Example

- **Feature extraction/engineering:** using domain knowledge to extract features from data.
- **Feature:** a piece of evidence intended to help the classifier map the input to the right target class
- **Feature vector:** a vector  $\vec{F}$ , the components  $F_j = \phi_j(d_j)$ , of which are results applying a feature function to the data point  $d_j$ .
- **Example:** “Spam vrs Ham” email?
  - number of “!” included in email body
  - length of the email in characters
  - occurrence of the word “cash” in the title or body.
- **Example feature vectors:**
  - (2, 2392, no) → HAM (genuine e-mail)
  - (4, 520, yes) → SPAM
  - (1, 2392, no) → HAM
  - (0, 16337, no) → HAM
  - (0, 61320, yes) → SPAM

# Feature Learning

- **Feature learning:** automatically extract features from data - replaces manual feature engineering that is prone to human errors.
- The use of **Supervised** and **Unsupervised learning** techniques for feature learning.
  - Supervised feature learning learns from labelled data and unsupervised learn from unlabelled data.
  - **GloVe vectors** [3] and **word2vec** [2] are vector representations for words that are learned using an unsupervised algorithm.
  - **BERT** [1] is a language model that produces contextual features (i.e., the same word has different representations in different contexts).
  - **Example:** She will park the car so we can walk in the park.
  - **Neural networks** aim to map the input to the output; the hidden-layers of these networks fine-tune input features automatically.
- **Principal Component Analysis** for dimensionality reduction

# Fundamental Methods Used for IE I

## Rule-Based

- **Rule-based:** Human experts (computational linguists) write general linguistic rules and task-specific extraction rules. Example, trigger keywords, regular expressions and patterns. Rule-based rules are language dependent, suffer from human ingenuity, time consuming, difficult to adapt to changes.

### Regular Expression Example

```
import re
sentence = "My sheffield email address is
            f.surname@sheffield.ac.uk and my
            personal email is f.surname@gmail.com"
reg_exp = "\S+@\S+"
emails = re.findall(reg_exp, sentence)
```

### Downside

- Write a regular expression to extract only sheffield or gmail addresses
- Know all the possible exact words or email domain to extract specific email addresses
- Considered as a brute force rather than a learning approach.

# Machine Learning Methods for IE

Machine-learning based

- **Machine Learning based (supervised):**

- Humans (domain experts) manually annotate text spans indicating entities, relations, facts, etc. in a training corpus;
- features are manually or automatically engineered (or a mixture of the two, e.g. using neural networks and dependency trees);
- these are used to extract information that statistically correlates with classes of entities, relations, etc.

# Machine Learning Methods for IE

Machine-learning based

- Hidden Markov Models (HMMs)
  - a statistical model for sequence modelling.
  - assigns a joint probability to paired word sequence  $X = X_1 \dots, X_n$  and label sequence  $Y = \{Y_1 \dots, Y_n\}$ .<sup>3</sup>
  - parameters are trained to maximize the joint likelihood, i.e.,  $\arg \max_Y P(Y, X) = \arg \max_Y P(Y|X)P(X)$ .
  - HMMs tries to model  $P(X)$  while learning the pattern
- Conditional Random Fields (CRF)
  - focuses on modelling  $P(Y|X)$ , which means  $P(X)$  can be modelled using a feature function of choice.
  - requires fewer parameters to learn
  - the linear chain CRF models the dependencies among adjacent labels.
- Support Vector Machines (SVMs) and Softmax Function
- Artificial Neural Networks (NNs), in particular “deep” neural nets for sequence tagging (RNN, LSTM), CNN, GCN

---

<sup>3</sup>In literature, observed features are typically denoted as  $Y$  while hidden variables are  $X$ .

# Experimental Methodology

- Writing an exam as an analogy for experimental methodology.
- Create a **gold data** - set of reference corpus with **ground truth**
- Split gold data into three parts:
  - **development/training set:** used to study the data, train machine learning processes; can be inspected
  - **development test** ("devtest") set: cannot be inspected, cannot be used for training; repeatedly used to measure improvements of system quality by comparing system output with ground truth.
  - **test set:** cannot be inspected; only used once for final evaluation run at project end. Completely **unseen data** (to the system and developers).
- Gold data split:  
could be e.g. 80% train: 10% dev-test : 10% test
- What happens when the exam is on a different subject?

# Named Entity Recognition (NER)

**NER:** Identify entities (such as person (PER), organisation (ORG), location (LOC), miscellaneous (MISC)) by assigning tags to chunks in text.

## BIO Encoding:

- **Text:** "Kungliga Hovkapellet is a Swedish orchestra, originally part of the Royal Court in Sweden's capital Stockholm"
- Annotate word tokens with beginning (B), inside(I)/outside(O) of class information:
- **Kungliga\_B-ORG** Hovkapellet\_I-ORG  
is\_O a\_O  
**Swedish\_B-MISC**  
orchestra\_O ,\_O originally\_O part\_O of\_O the\_O  
**Royal\_B-ORG** Court\_I-ORG  
in\_O  
**Sweden\_B-LOC**  
capital\_O **Stockholm\_B-LOC**
- Other encoding schemes include BIOES - ..., End, Single entity

# NER Libraries

<http://corenlp.run/>



version 4.4.0

— Text to annotate —

Kungliga Hovkapellet is a Swedish orchestra, originally part of the Royal Court in Sweden's capital Stockholm

— Annotations —

parts-of-speech  named entities  dependency parse

— Language —

English

Submit

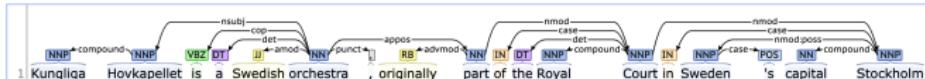
## Part-of-Speech:

1 NNP NNP VBD DT IN NN RB NN IN DT NNP NNP IN NNP POS NN NNP  
Kungliga Hovkapellet is a Swedish orchestra , originally part of the Royal Court in Sweden 's capital Stockholm

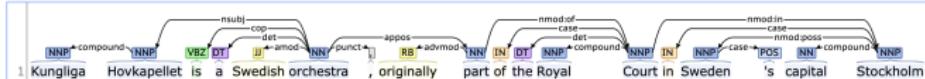
## Named Entity Recognition:

1 PERSON NATIONALITY ORGANIZATION COUNTRY CITY  
Kungliga Hovkapellet is a Swedish orchestra , originally part of the Royal Court in Sweden 's capital Stockholm

## Basic Dependencies:



## Enhanced++ Dependencies:



# NER Libraries

← → 🔍 https://huggingface.co/html/ 🌐

HMTL for NLP

Kungliga Hovkapellet is a Swedish orchestra, originally part of the Royal Court in Sweden's capital Stockholm

Q

[Named Entity Recognition](#) ↗

Kungliga Hovkapellet PERSON is a Swedish NNP orchestra, originally part of the Royal Court ORG in Sweden GPE 's capital Stockholm GPE

[Entity Mention Detection](#) ↗

Kungliga Hovkapellet PER is a Swedish GPE orchestra ORG , originally part ORG of the Royal Court ORG in Sweden GPE 's capital GPE Stockholm GPE

[Relation Extraction](#) ↗

Kungliga Hovkapellet is a Swedish OSU orchestra SUBJ , originally part SUBJ SUBJ of the Royal Court SUBJ OSU In Sweden's capital Stockholm OSU

```
graph LR; A[Swedish OSU] -- GEN-AFF --> B[orchestra SUBJ]; B -- PART-WHOLE --> C[originally part SUBJ SUBJ]; C -- PART-WHOLE --> D[the Royal Court SUBJ OSU]; D -- GEN-AFF --> E[Stockholm OSU]
```

# NER Libraries

https://explosion.ai/demos/displacy-ent?text=Kungliga%20Hovkapellet%20is%20a%20Swedish%20orchestra%2C%20originally%20part%20of%20the%20Royal%20Court%20in%20Sweden's%20capital%20Stockholm

About us Software & Demos

## displaCy Named Entity Visualizer

Kungliga Hovkapellet is a Swedish orchestra, originally part of the Royal Court in Sweden's capital Stockholm

Model ? English - en\_core\_web\_sm (v3.1.0)

Entity labels (select all)

<input checked="" type="checkbox"/> PERSON	<input checked="" type="checkbox"/> NORP	<input checked="" type="checkbox"/> ORG	<input checked="" type="checkbox"/> GPE	<input checked="" type="checkbox"/> LOC
<input checked="" type="checkbox"/> PRODUCT	<input type="checkbox"/> EVENT	<input type="checkbox"/> WORK OF ART		
<input type="checkbox"/> LANGUAGE	<input checked="" type="checkbox"/> DATE	<input type="checkbox"/> TIME	<input type="checkbox"/> PERCENT	
<input type="checkbox"/> MONEY	<input type="checkbox"/> QUANTITY	<input type="checkbox"/> ORDINAL	<input type="checkbox"/> CARDINAL	

Kungliga Hovkapellet PERSON is a Swedish NORP orchestra, originally part of the Royal Court ORG in Sweden GPE's capital Stockholm GPE

# BiLSTM Approach to NER

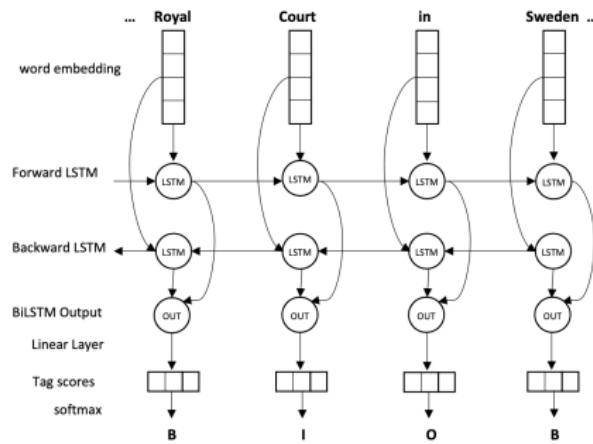


Figure: BiLSTM-Softmax

- BiLSTM-Softmax disregards label dependencies and simply feeds the tag scores into a softmax layer to get the label classification.
- BiLSTM-CRF is more expressive, considers label dependencies, suitable for complex sequence labelling problems such as fine-grained NER (i.e., tags such as {B-ORG,I-ORG} to identify “Royal Court”).

# Relation Extraction (RE)

"[1] Kungliga Hovkapellet is a Swedish orchestra, originally part of the Royal Court in Sweden's capital Stockholm. [2] The orchestra originally consisted of both musicians and singers. [3] It had only male members until 1727, when Sophia Schröder and Judith Fischer were employed as vocalists; in the 1850s, the harpist Marie Pauline Åhman became the first female instrumentalist. [4] From 1731, public concerts were performed at Riddarhuset in Stockholm. [5] Since 1773, when the Royal Swedish Opera was founded by Gustav III of Sweden, the Kungliga Hovkapellet has been part of the opera's company."

# Relation Extraction (RE)

"[1] Kungliga Hovkapellet is a Swedish orchestra, originally part of the Royal Court in Sweden's capital Stockholm. [2] The orchestra originally consisted of both musicians and singers. [3] It had only male members until 1727, when Sophia Schröder and Judith Fischer were employed as vocalists; in the 1850s, the harpist Marie Pauline Åhman became the first female instrumentalist. [4] From 1731, public concerts were performed at Riddarhuset in Stockholm. [5] Since 1773, when the Royal Swedish Opera was founded by Gustav III of Sweden, the Kungliga Hovkapellet has been part of the opera's company."

- **Relation Extraction:** involves the detection and classification of semantic relationships between pairs of entity mentions.
- **Example 1:** To identify the relation fact (Stockholm, capital, Sweden) - Evidence in [1]
- **Example 2:** (Riddarhuset, country, Sweden) - Evidence in [1] and [4]

# RE: Representation Learning

## Input Features

### Text:

[1] Kungliga Hovkapellet is a Swedish orchestra, originally part of the Royal Court in Sweden's capital Stockholm.

### Representation Learning:

- **Glove pre-trained embeddings:** to represent words, i.e., word  $w_i$  is mapped to  $\mathbf{w}_i$ ;
- **Relative position embeddings:** to get the relative position of each word to the subject and object entities.
  - the relative position of "capital" in the text to "Sweden" and "Stockholm" are 1 and -1, respectively.
  - the relative position of "orchestra" in the text to "Sweden" and "Stockholm" are -7 and -9, respectively.
  - each relative position  $p$  is mapped to a vector  $\mathbf{d}^p$ .
  - relative position embeddings for "orchestra" is a concatenated vector  $\mathbf{d}_i = [\mathbf{d}^{-7}; \mathbf{d}^{-9}]$ .
- **Input embeddings:** concatenation of Glove and Position Embeddings, i.e.,  $\mathbf{e}_i = [\mathbf{w}_i; \mathbf{d}_i]$ . That is, for the text, we have  $\mathbf{E} = \{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_n\}$

# RE: Modelling

## Sequence-based methods

### 1. Sequence-based methods (i.e., BiLSTM, CNN) [6]

- BiLSTM can model **long distance relationships** (e.g., relationship between **Kungliga Hovkapellet** and **Stockholm**).
- **BiLSTM with attention mechanisms** to capture important features with respect to the relation target. [8] (e.g., “capital” is the most important word to determine the relationship between **Sweden** and **Stockholm**)
- **CNNs** for sequence modelling [5].

### 2. **Downside:** existence of multiple entity pairs in the text, resulting in a mixture of relations in the final representation causing confusing clues for classification.

# RE: Modelling

## Graph-based methods

**Graph-based methods:** a class of neural networks that learn representations for graphs (e.g., Graph Convolutional Network (GCN)).

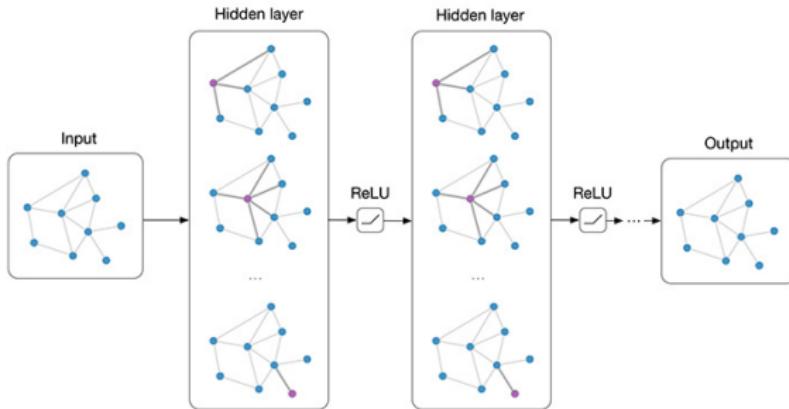


Figure: Multi-layer GCN with first-order filters.<sup>4</sup>

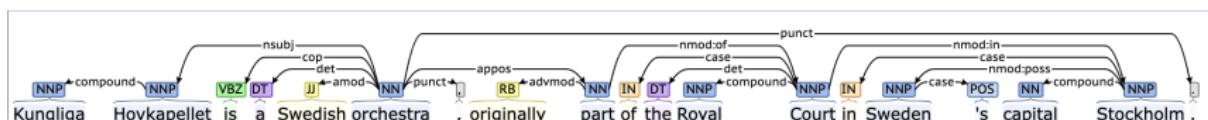
- Each node is represented as a vector
- message is transported from neighbor nodes to learn node representations.
- Retains the original structure of the graph

<sup>4</sup> Image from <https://tkipf.github.io/graph-convolutional-networks/>

# RE: Modelling

## Graph-based methods

### Dependency tree/ syntax structure/ syntax graph:

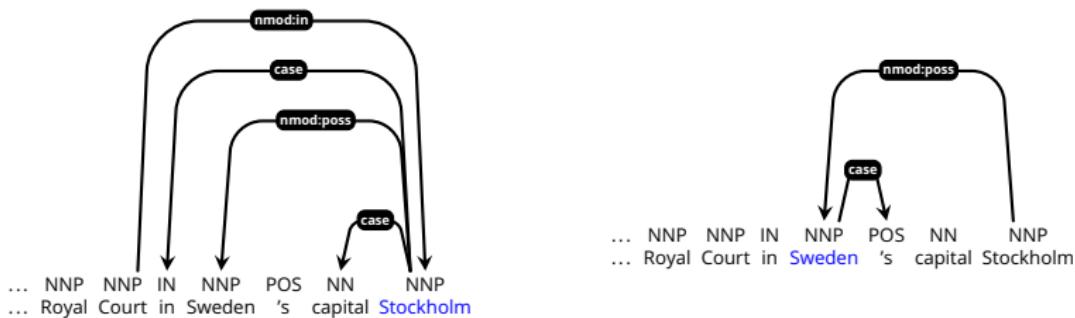


- **Goal:** Select only salient features that are connected to the target relation by applying a GCN on syntactic structures.
  - words can be represented by  $\mathbf{H} = BiLSTM(\mathbf{E})$
  - GCN transfers information from neighbor words in the syntax graph

# RE: Modelling

## Graph-based methods

1. Prune irrelevant syntactic dependencies that do not contribute to the relation and apply a  $K$ -layer (small integer value  $K$ ) GCN on only the relevant features (somewhat rule-based) [7].
  - **relation cues** are typically **close to the entities** in the dependency tree.



- assumption that a **pre-defined  $K$**  fits all instances
- some relations are **distant from entity nominals** (e.g., (Kungliga Hovkapellet, member, Royal Court) -  $K$  can be large)
- **Solution:** applying an attention mechanism over several GCN layers [4]

# RE: Model Training

## Classification & Training Objective

- We have our final representation  $\mathbf{v}$

$$\mathbf{v} = \text{maxpool}(\text{GCN}(\text{BiLSTM}(\mathbf{E}))) \quad (1)$$

- $\mathbf{E}$  denote the input embeddings
- The relation prediction  $\hat{\mathbf{y}} = \{\hat{y}_1, \dots, \hat{y}_m\}$

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{v}\mathbf{W} + \mathbf{b}) \quad (2)$$

- we minimize the cross-entropy loss between the true relation label distribution and the predicted label distribution over all sentences.

$$\mathcal{L} = \sum_s \left( -\frac{1}{m} \sum_{i=1}^m y_i \log \hat{y}_i \right) \quad (3)$$

- $\mathbf{E}$  denote the input embeddings;  $\mathbf{W}, \mathbf{b}$  denote learnable weights and bias;  $m$  is the number of relations;  $\mathbf{y} = \{y_1, \dots, y_m\}$  is the one-hot represented ground truth.

# Question Answering

## An Application

### Text:

[1] Kungliga Hovkapellet is a Swedish orchestra, originally part of the Royal Court in Sweden's capital Stockholm.

### Relational Facts or Knowledge base:

- (Kungliga Hovkapellet, member\_of, Royal Court); (Kungliga Hovkapellet, country\_of\_origin, Sweden); (Kungliga Hovkapellet, city\_of\_origin, Stockholm); (Royal Court, city\_of\_headquarters, Stockholm); (Kungliga Hovkapellet, country\_of\_headquarters, Sweden); (Sweden, capital\_of, Sweden)

**Question:** In which country is the city Stockholm located?

- Detect the named entity **Stockholm** in the question.
- Detect the relation “city\_of” to get the partial fact (Stockholm, city\_of, ?)
- Find the closest fact in the knowledge base, in this case, (Stockholm, capital\_of, Sweden), and make a prediction.

# Question Answering

An Application

## **Question:**

When was the Swedish orchestra Kungliga Hovkapellet established?

# Question Answering

An Application

## Question:

When was the Swedish orchestra Kungliga Hovkapellet established?

The image shows a Google search results page. The search query is "When was the Swedish orchestra Kungliga Hovkapellet established?". The top result is a bolded blue link to a page stating the year 1526. Below this, a snippet of text provides historical context about the orchestra's establishment and evolution.

Google

When was the Swedish orchestra Kungliga Hovkapellet established?

All News Images Maps Videos More Tools

About 8,740 results (0.54 seconds)

**1526**

Kungliga Hovkapellet (The Royal Swedish Orchestra) has been at the heart of Swedish culture for almost 500 years. Founded in **1526** as the orchestra to the Royal Court, it became the orchestra of the Royal Swedish Opera in 1773.

# Summary

In this session, we learned:

- what information extraction is, and a bit of its history.
- the difference between rule-based and machine learning approaches
- how named entities and relations are extracted from text and its application to question answering.

# The End

Questions? Comments?

# References I

-  Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova.  
Bert: Pre-training of deep bidirectional transformers for language understanding.  
*arXiv preprint arXiv:1810.04805*, 2018.
-  Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean.  
Efficient estimation of word representations in vector space.  
*arXiv preprint arXiv:1301.3781*, 2013.
-  Jeffrey Pennington, Richard Socher, and Christopher D Manning.  
Glove: Global vectors for word representation.  
In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
-  Kai Sun, Richong Zhang, Yongyi Mao, Samuel Mensah, and Xudong Liu.  
Relation extraction with convolutional network over learnable syntax-transport graph.  
In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8928–8935, 2020.
-  Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, and Jun Zhao.  
Relation classification via convolutional deep neural network.  
In *Proceedings of COLING 2014, the 25th international conference on computational linguistics: technical papers*, pages 2335–2344, 2014.
-  Shu Zhang, Dequan Zheng, Xinchen Hu, and Ming Yang.  
Bidirectional long short-term memory networks for relation classification.  
In *Proceedings of the 29th Pacific Asia conference on language, information and computation*, pages 73–78, 2015.

# References II



[Yuhao Zhang, Peng Qi, and Christopher D Manning.](#)

Graph convolution over pruned dependency trees improves relation extraction.  
*arXiv preprint arXiv:1809.10185*, 2018.



[Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu.](#)

Attention-based bidirectional long short-term memory networks for relation classification.  
In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*, pages 207–212, 2016.



# Sequence to Sequence Models, Attention and the Transformer with applications to Neural Machine Translation

---

Dr Harish Tayyar Madabushi

[h.tayyarmadabushi@sheffield.ac.uk](mailto:h.tayyarmadabushi@sheffield.ac.uk)

<https://www.harishtayyarmadabushi.com/>

# Previously Covered

- **Task:** Neural Language Modelling
  - Estimate the probability of a text (for example, a sentence)
- **Model:** Recurrent Neural Networks (RNNs)
  - Given a sequence of tokens (for example, words in a sentence), RNNs “read” them one by one and process the information
- **Improvement:** Attention
  - At different steps, let a model “focus” on different parts of the input

# In this Lecture

- **Sequence to Sequence Models**
- **Demonstrating Task:** Neural Machine Translation (NMT)
- **Model:** Encoder-Decoder with RNNs
- **Improvement:** Attention and The Transformer
- **Evaluation:** Automatic Metrics and Manual Judgements

# Sequence to Sequence Models

- Sequence to Sequence models have revolutionised NLP applications and are now commonly used in end user applications.
- What are Sequence to Sequence models?
  - Input is a Sequence (any sequence)
  - Output is a Sequence (again, any sequence, but for our purpose: text)

# Sequence to Sequence Models: Examples

Thoughts?

# Sequence to Sequence Models: Examples

Machine Translation	A pesquisa em PNL é muito excitante.	NLP research is very exciting.
Speech to Text		Hello there!
Image Captioning		A woman playing tennis
NER	It is our choices, Harry, that show what we truly are, far more than our abilities.	Harry
Neural Music Generation	No input (or some note)	

# Sequence to Sequence Models: More Examples

Who is wearing glasses?



→ The Man



→ The Woman

# Sequence to Sequence Models: More Examples\*

## Logical Inference

**Input:** Shelley is from Virginia, but is visiting that city with that famous market where they throw the fish. Going home next Tuesday!

Question: Is it likely that Shelley will be near the Pacific Ocean this weekend?

**Model Output:** The city with the famous market where they throw the fish is Seattle, Washington. Seattle is on the Pacific Ocean. Shelley is visiting Seattle, so she will be near the Pacific Ocean this weekend. The answer is "yes", it is likely that Shelley will be near the Pacific Ocean this weekend.

\*PaLM: MoE Decoder-only



# Demonstrating Task Neural Machine Translation

# What is Machine Translation?

**Thoughts?**

# What is Machine Translation?

**Machine Translation** (MT) consists of translating a sentence  $x$  from one language (the *source*) to a sentence  $y$  in another language (the *target*)



# What is *Neural* Machine Translation?

- **Neural Machine Translation** (NMT) is Machine Translation using neural networks (as opposed to alignment and phrase based translation).
- Typically we use **sequence-to-sequence models** (seq2seq).
- These models are **end-to-end differentiable**

# Task Formulation

- We want to translate from English to Spanish
- Formally, what this means is: we want to find best Spanish sentence  $y$ , given English sentence  $x$

**Hello, how are you** → **Hola como estas**

→ **Hola como estamos**

→ **Dónde estás**

# Task Formulation

- We want to translate from English to Spanish
- Formally, what this means is: we want to find best Spanish sentence  $y$ , given English sentence  $x$
- Problems:
  - How do we come up with candidate sentences?
  - How do we search through this large list of possible sentences?

# Task Formulation

- Problems:
  - How do we come up with candidate sentences?
    - Generate sentences using a neural network (seq2seq)
    - We are going to teach the neural network to generate candidate sentences
  - How do we search through this large list of possible sentences?
    - Beam Search

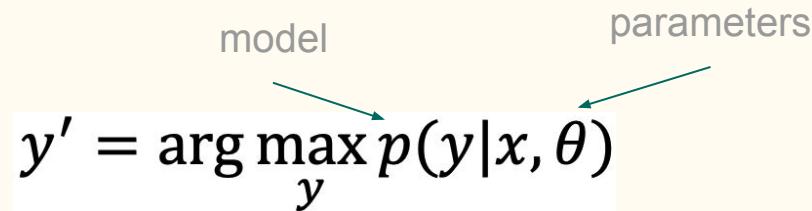
# Task Formulation

More Formally:

- We want to find best Spanish sentence  $y$ , given English sentence  $x$

$$y' = \arg \max_y p(y|x, \theta)$$

model                            parameters



**modelling**

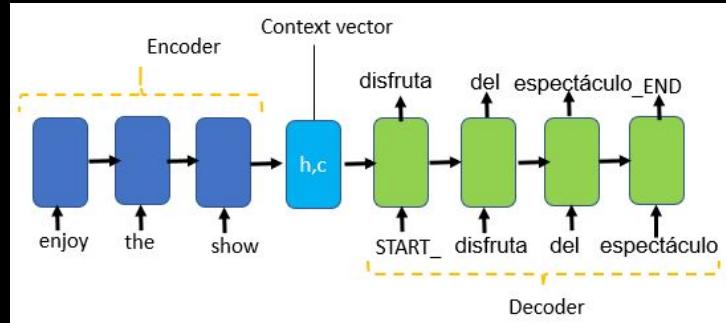
What is the model  
 $p(y|x, \theta)$ ?

**learning**

How do we find to  
find  $\theta$

**search**

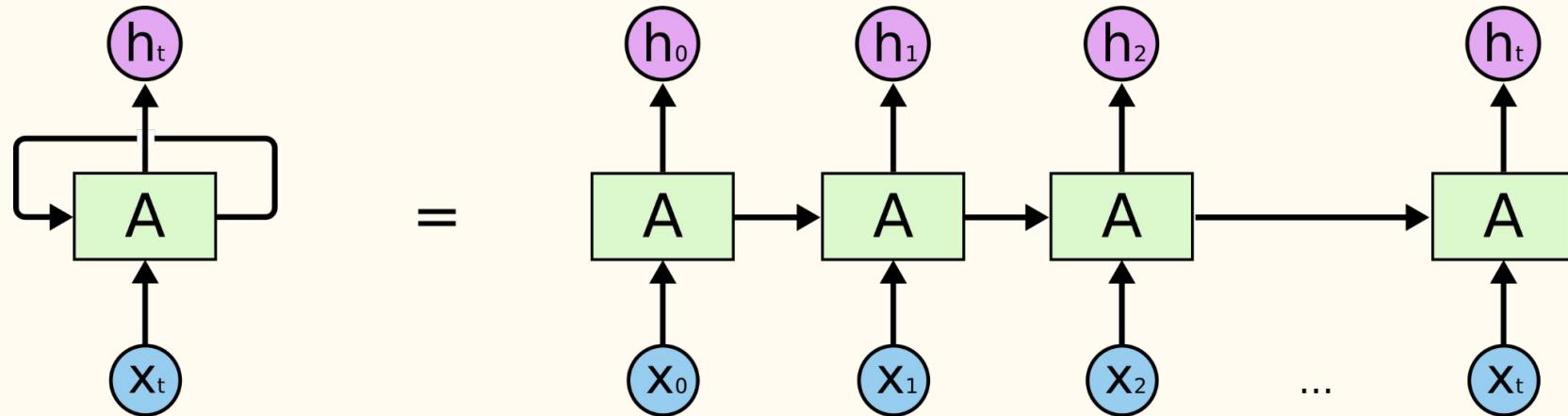
How do we find the  
*best*?



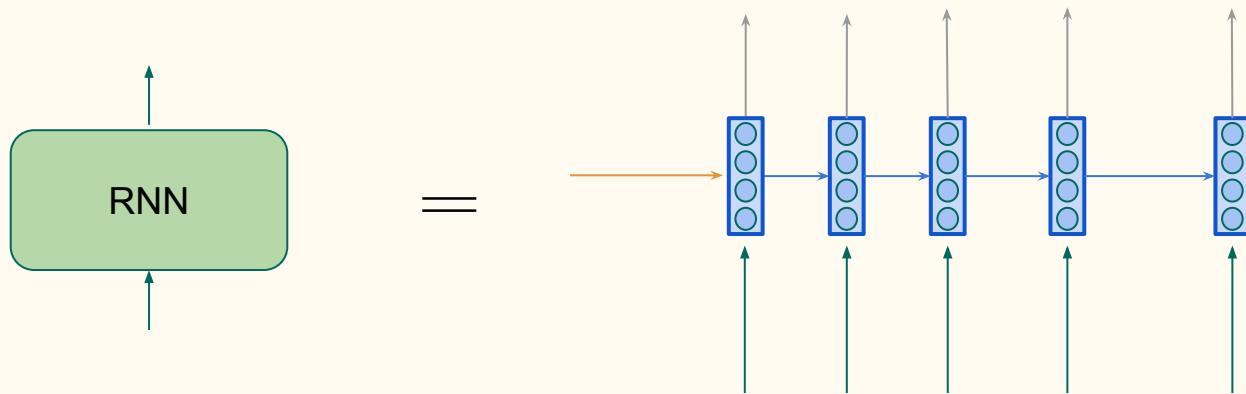
# The Model

## Encoder-Decoder Architecture (with RNNs)

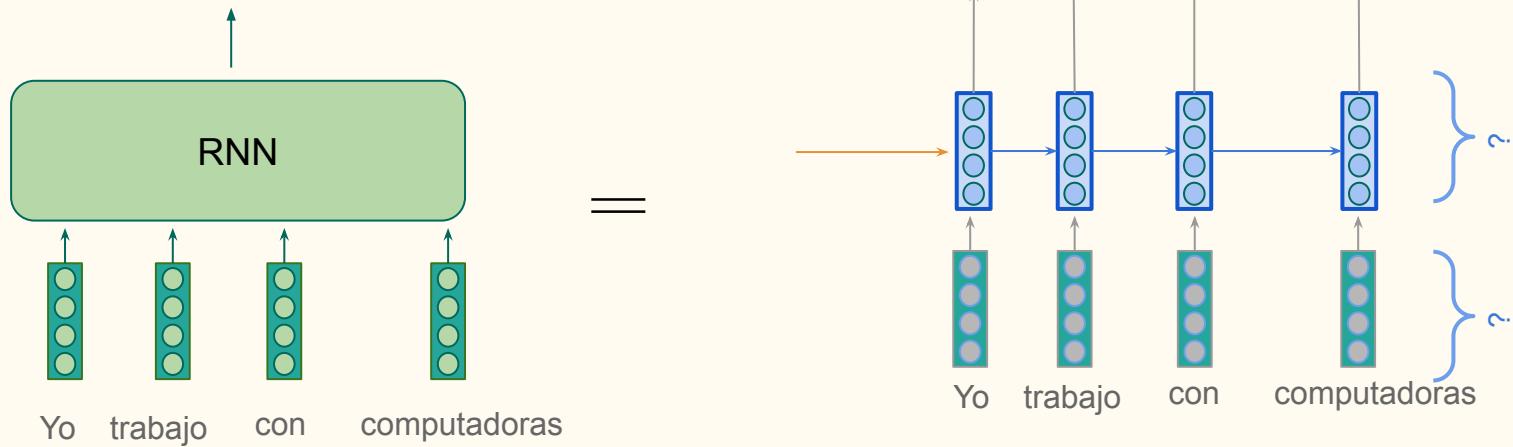
# Quick Note on RNN representations



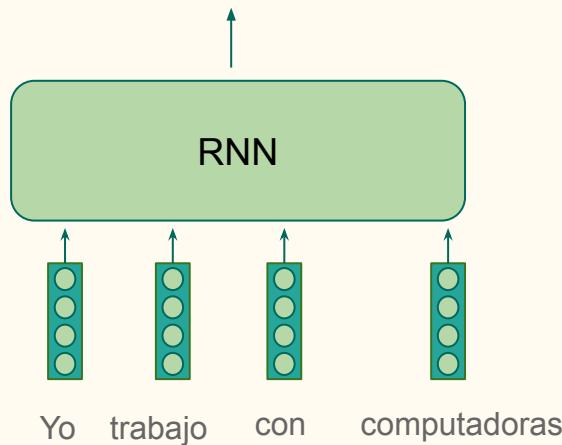
# Quick Note on RNN representations



# Quick Note on RNN representations

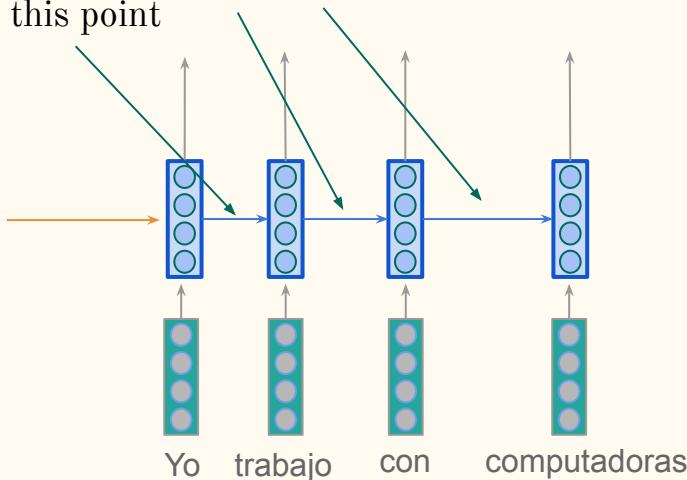


# Quick Note on RNN representations



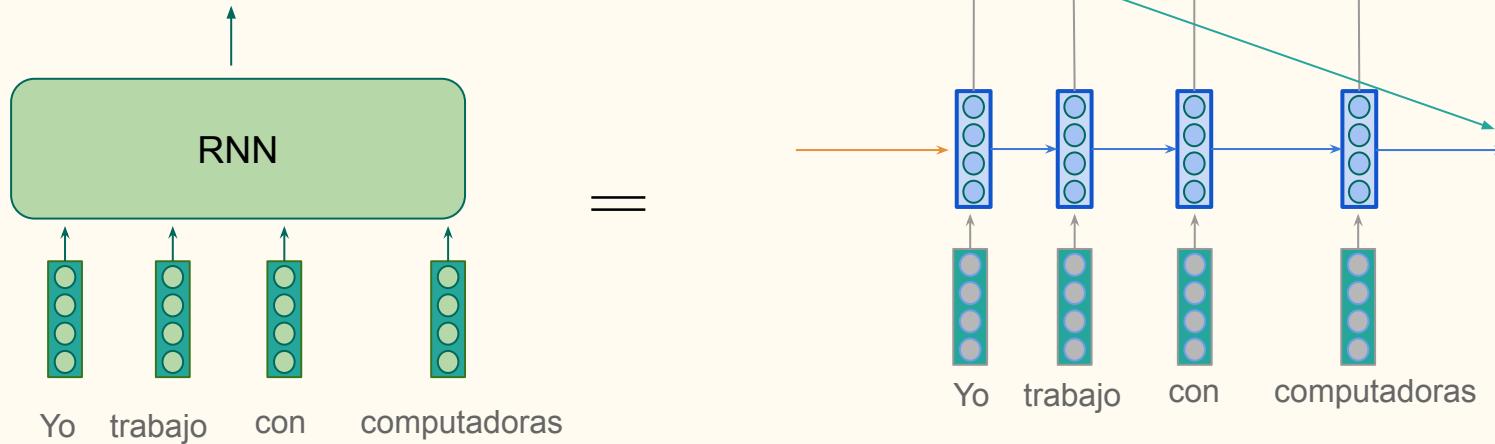
=

“Representation” of sentence  
up to this point

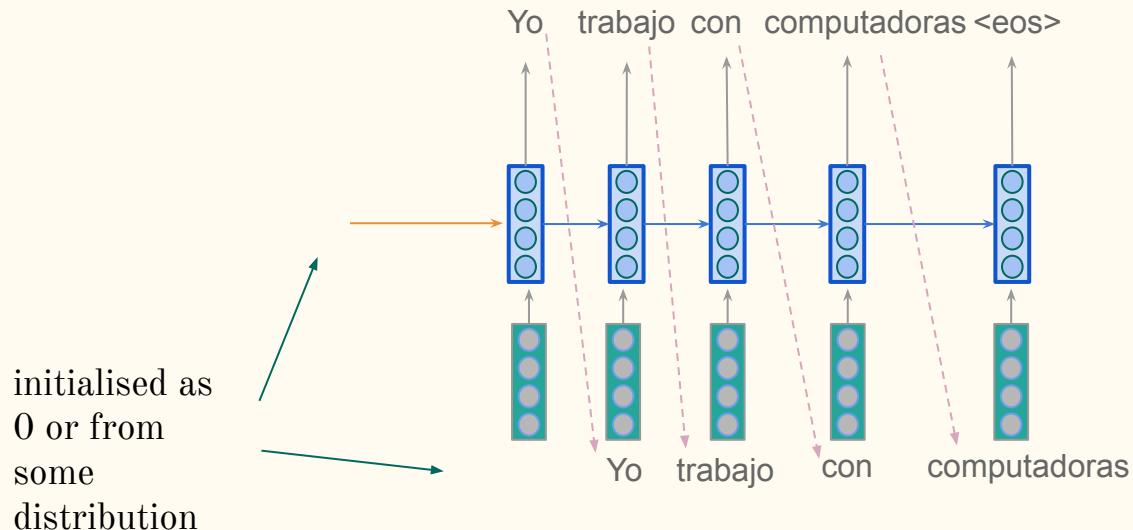


# Quick Note on RNN representations

“Representation” of entire sentence.



# Recall: Language Modelling with RNNs



Notice that RNN based Language Models provide contextual representations

Peters et al 2017

Peters et al 2018 (ELMo)

\* ELMo includes CNNs

They also capture grammatical information

Colorless green recurrent networks dream hierarchically

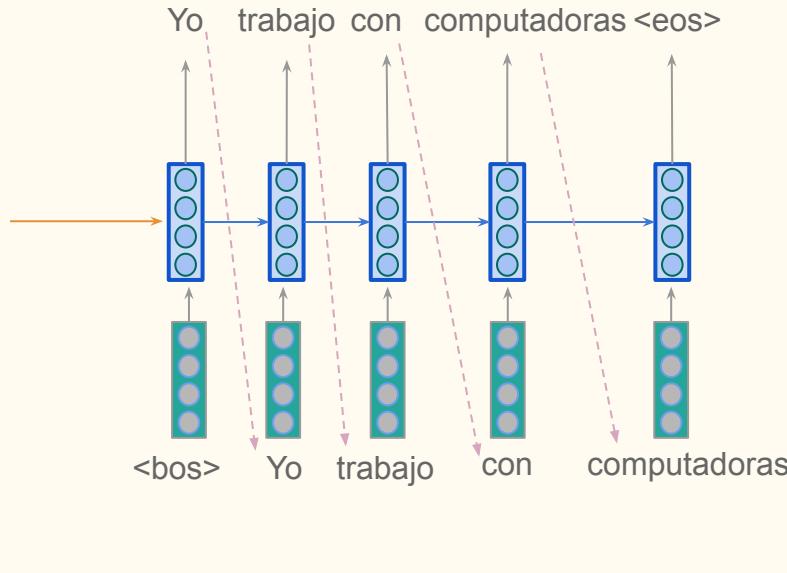
“the girl[you met yesterday through her friends] thinks.  
...”.

# Recall: Language Models

Language Models:

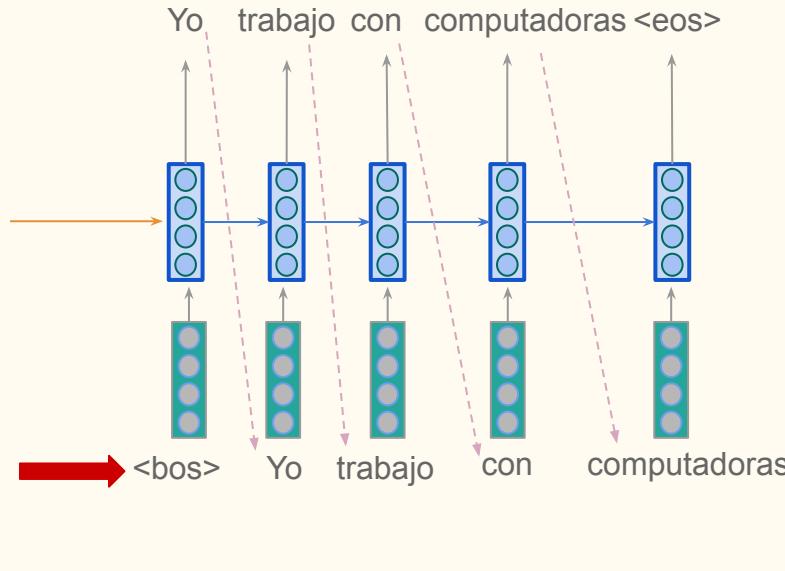
$$P(y_1, y_2, \dots, y_n) = \prod_{t=1}^n p(y_t | y_{<t})$$

# Language Modelling to Sequence Generation



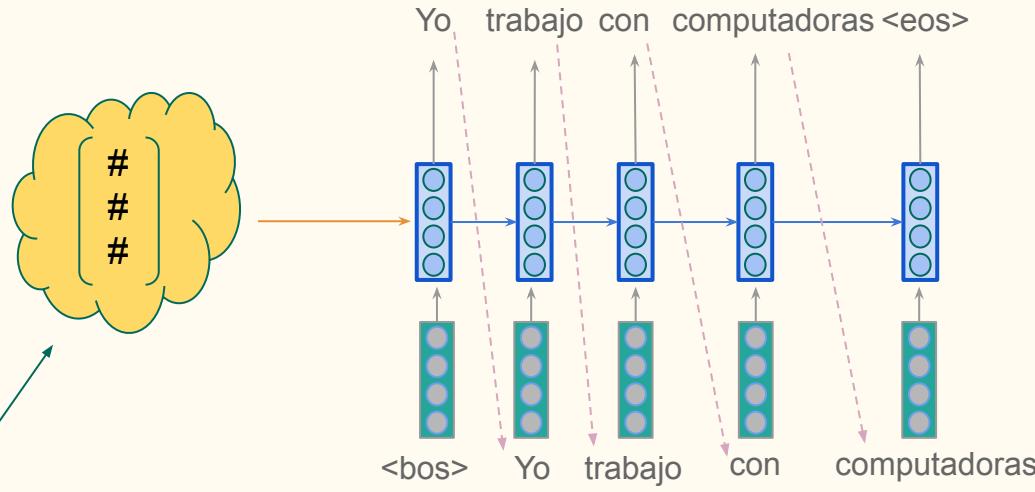
We can initialise “this” with a “prompt” that acts as the start of an idea or a thought that will now be fleshed out by the generated text

# Language Modelling to Sequence Generation



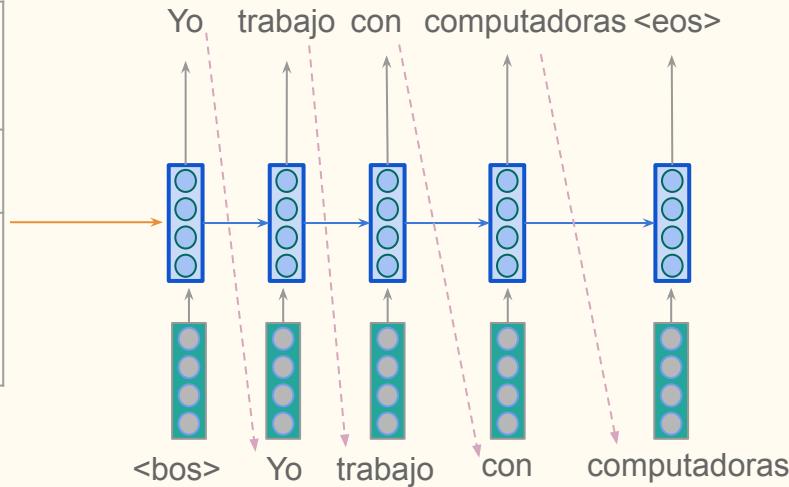
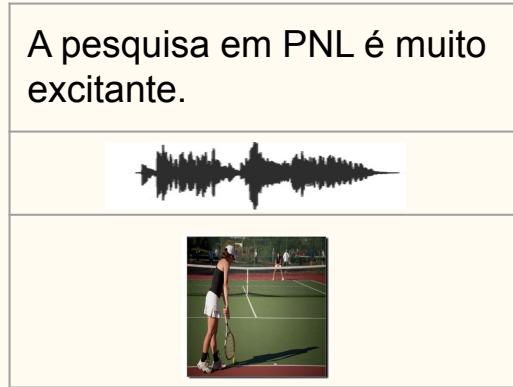
We can initialise “this” with a “prompt” that acts as the start of an idea or a thought that will now be fleshed out by the generated text

# Language Modelling to Sequence Generation



We can initialise this ***with a representation of an idea or a thought*** that will now be fleshed out by the generated text

# Language Modelling to Sequence Generation



We can initialise this ***with a representation of an idea or a thought*** that will now be fledged out by the generated text

# Encoding Information

A pesquisa em PNL é muito excitante.

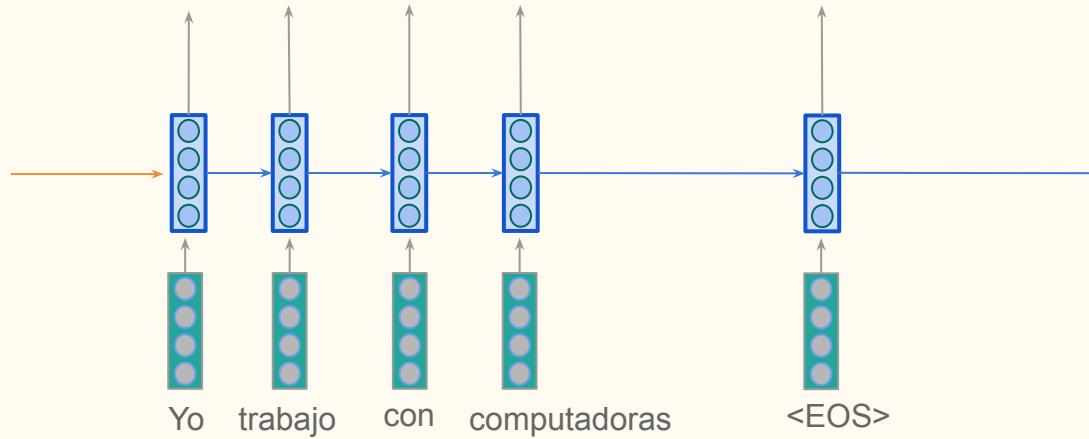


*How can we convert this into an “encoding” or a vector?*

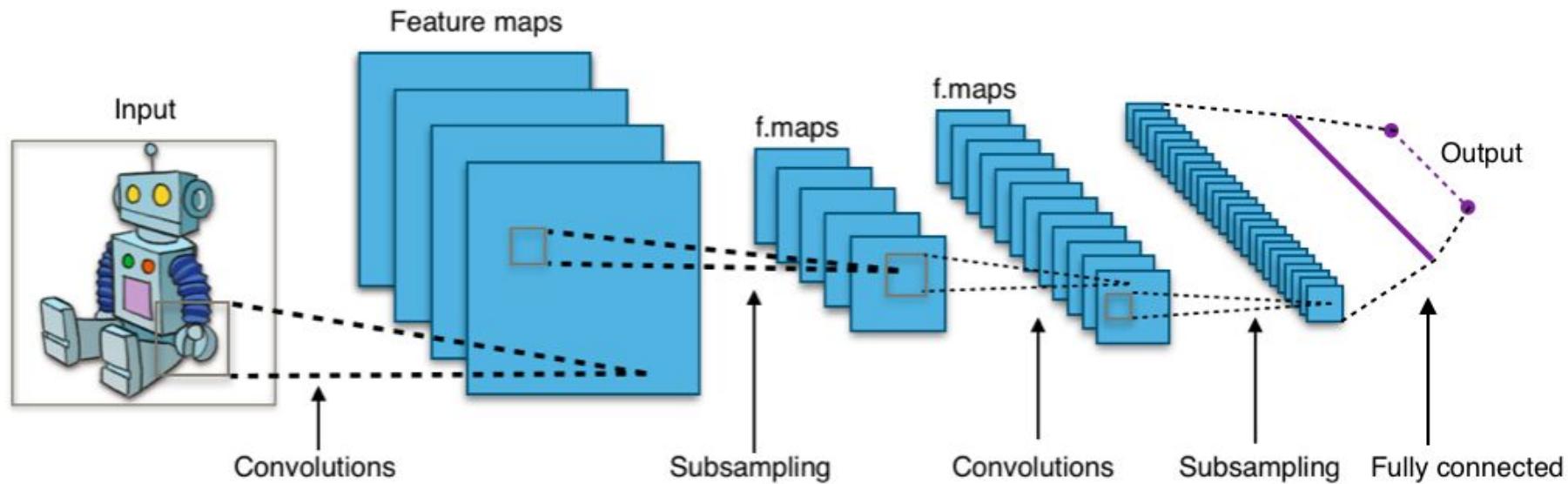
Let's start with the text?

# Encoding Information (text)

We already  
know how to  
do this!

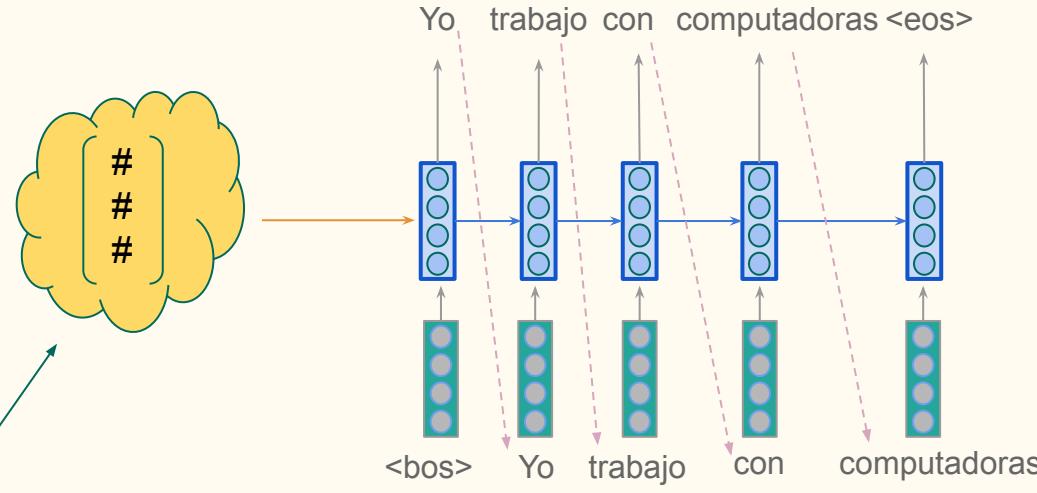


# Encoding Information (images)



# “Thought” vector to Encoder-Decoder

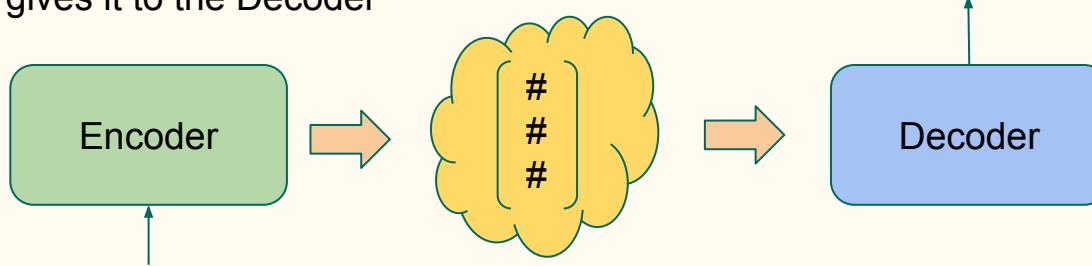
Use an  
Encoder!



We can initialise this ***with a representation of an idea or a thought*** that will now be fleshed out by the generated text

# Encoder-Decoder

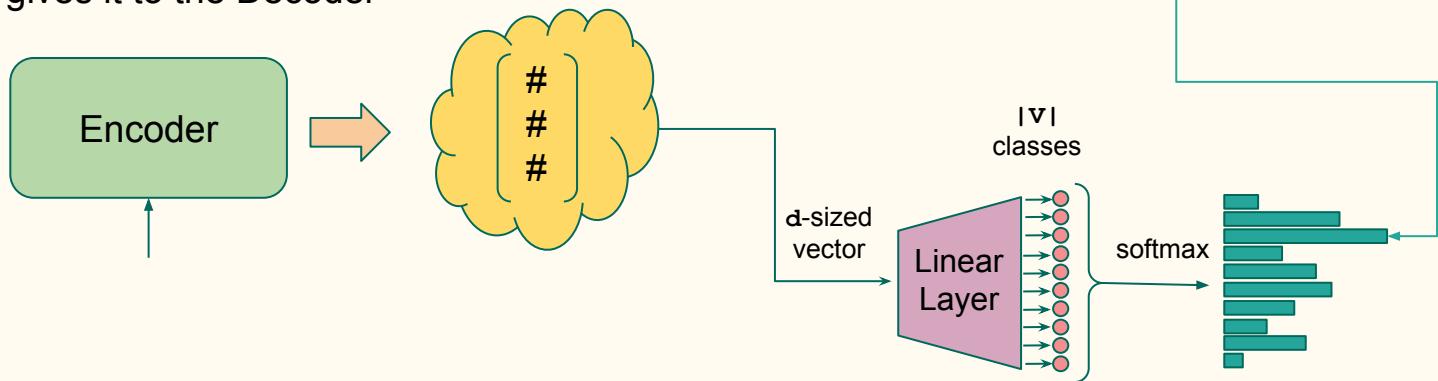
**Encoder** reads the *source*,  
produces its representation,  
and gives it to the Decoder



**Decoder** uses the *source*  
representation to generate  
the *output*

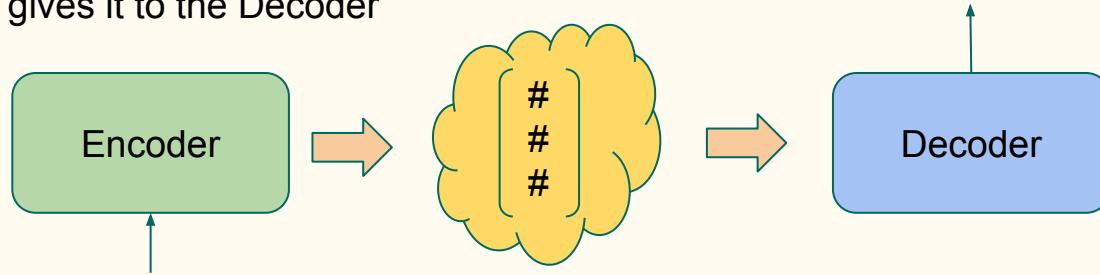
# Encoder to Classification (aside)

**Encoder** reads the *source*,  
produces its representation,  
and gives it to the Decoder



# Back to the Encoder-Decoder

**Encoder** reads the *source*,  
produces its representation,  
and gives it to the Decoder



**Decoder** uses the *source*  
representation to generate  
the *output*

# Conditional Language Models

Language Models:

$$P(y_1, y_2, \dots, y_n) = \prod_{t=1}^n p(y_t | y_{<t})$$

**Conditional**  
Language Models:

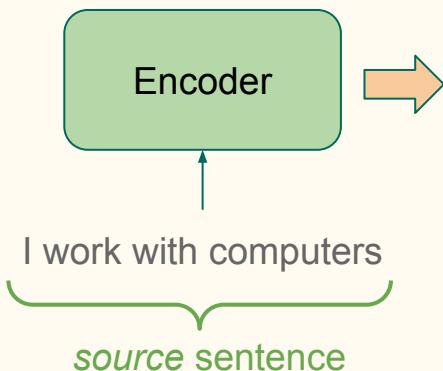
$$P(y_1, y_2, \dots, y_n, | \textcolor{brown}{x}) = \prod_{t=1}^n p(y_t | y_{<t}, \textcolor{brown}{x})$$

condition on source  $\textcolor{brown}{x}$

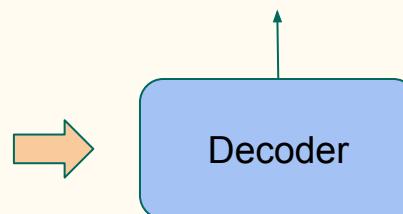
$\textcolor{brown}{x}$  does not always need to  
be a sequence of tokens.  
(e.g. image captioning)

# Encoder-Decoder for Translation

**Encoder** reads the *source*, produces its representation, and gives it to the Decoder

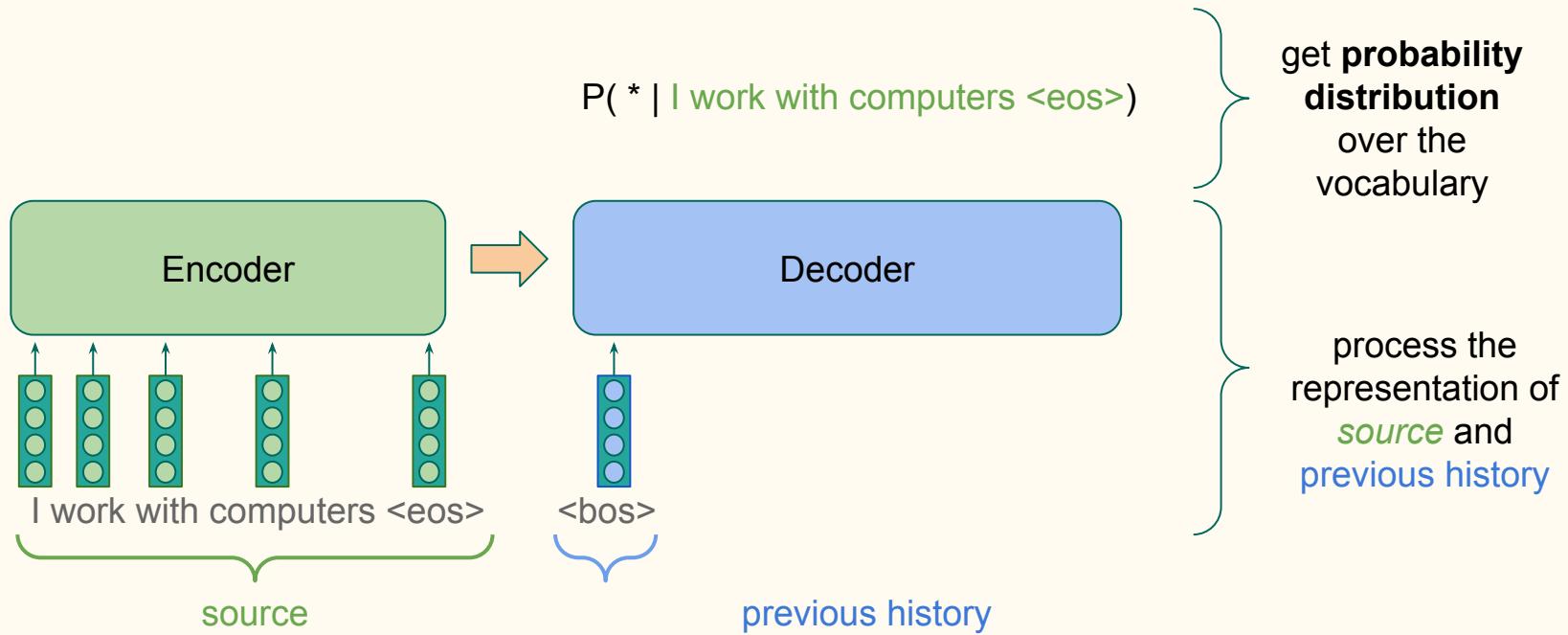


*target sentence*  
Yo trabajo con computadoras

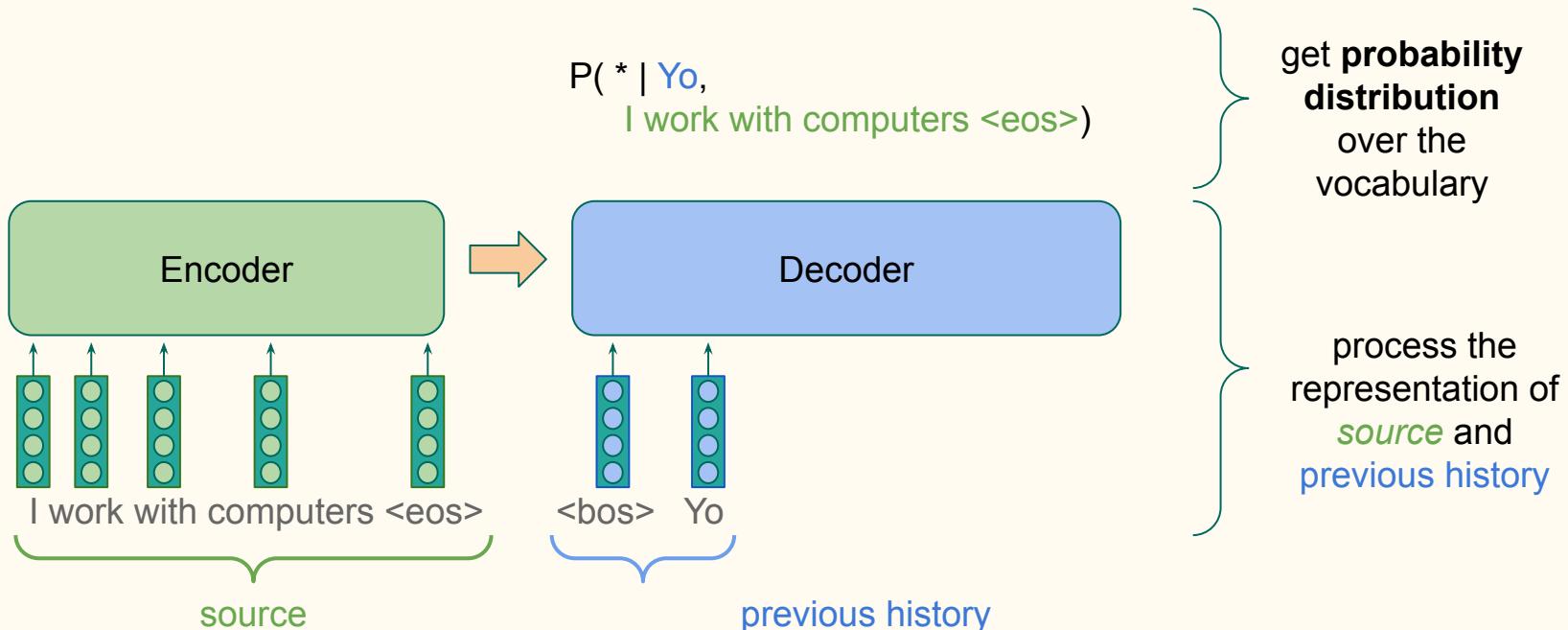


**Decoder** uses the *source* representation to generate the *target sentence*

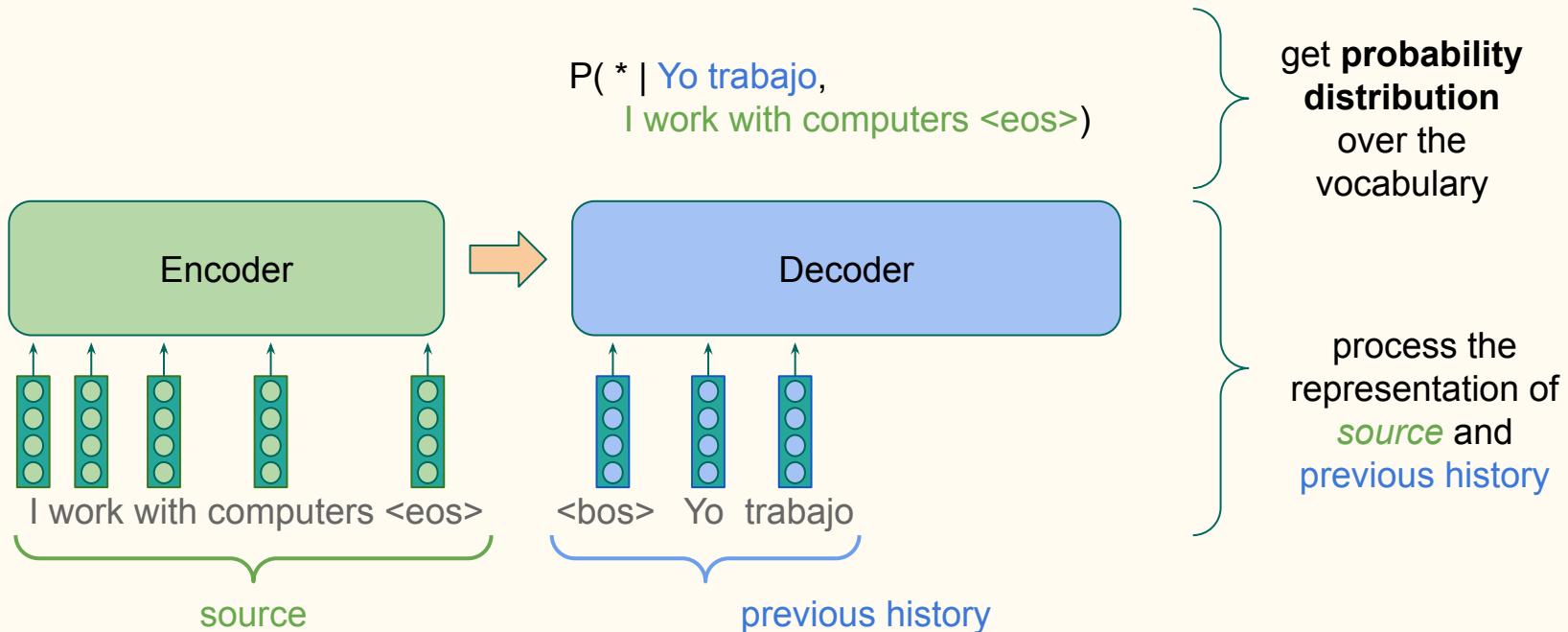
# Sequence Generation



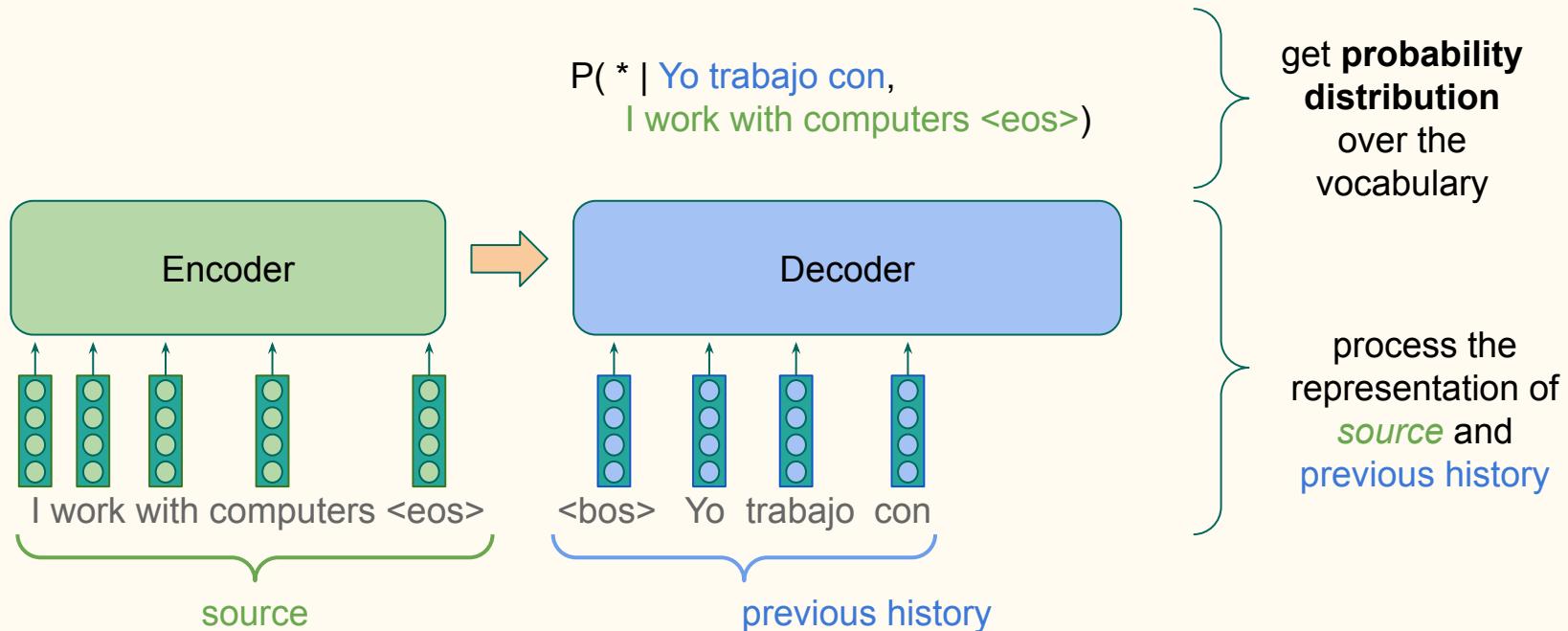
# Sequence Generation



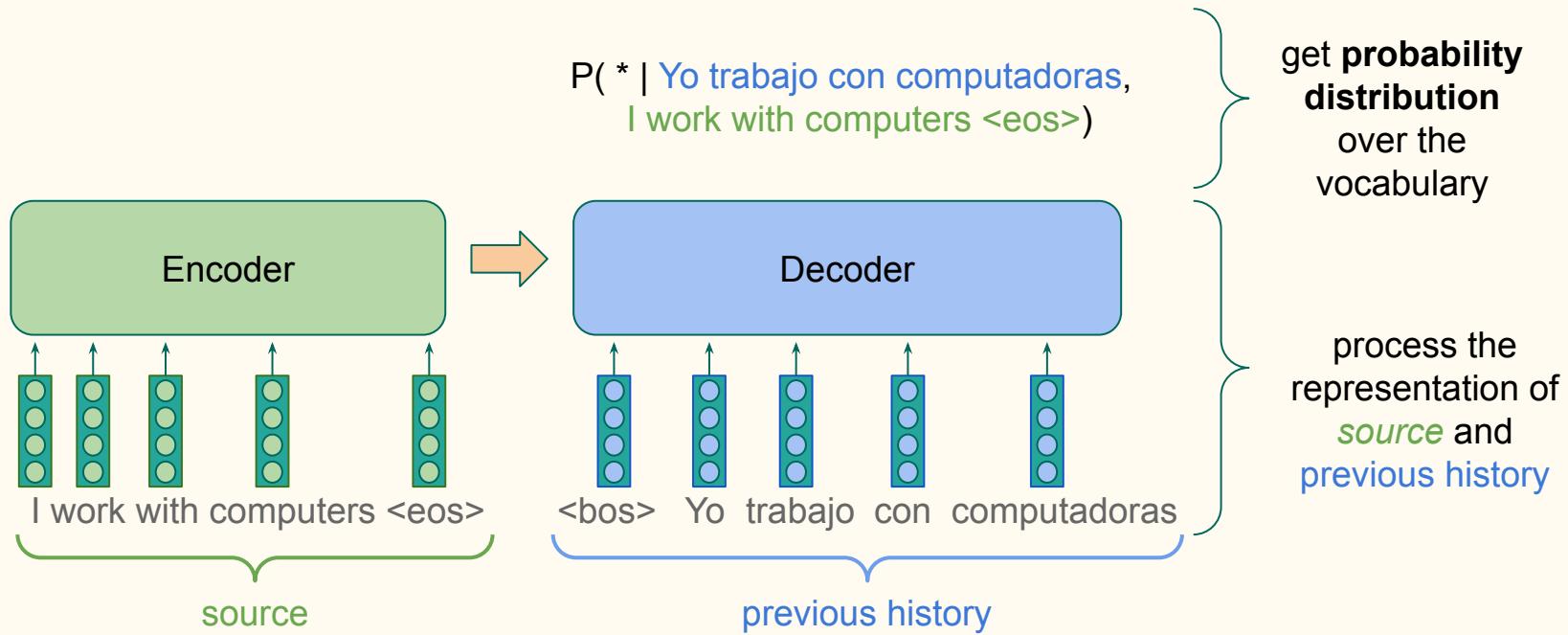
# Sequence Generation



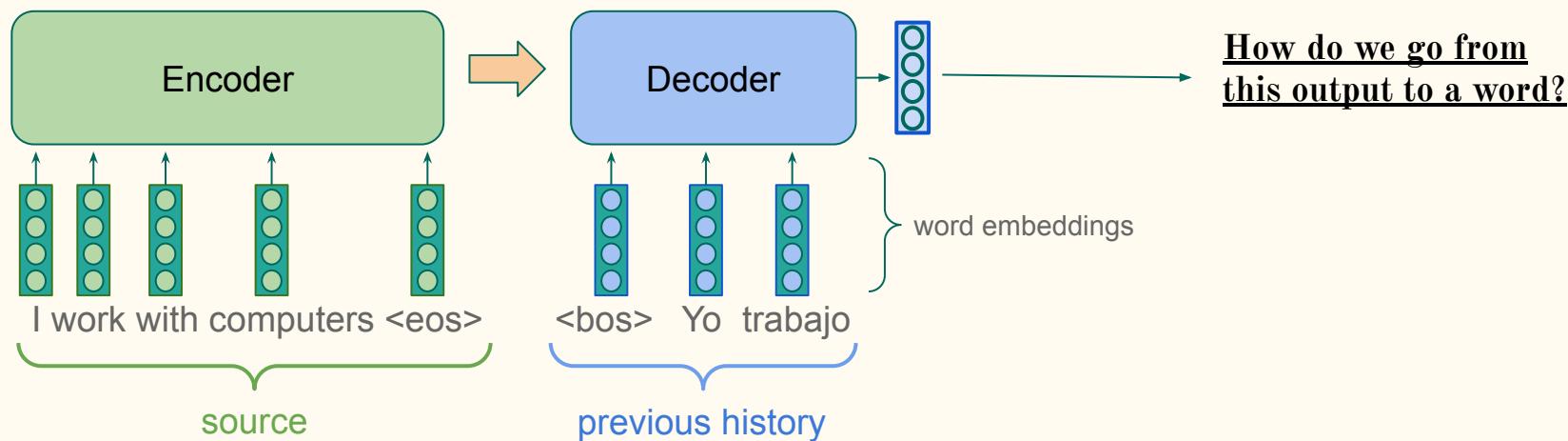
# Sequence Generation



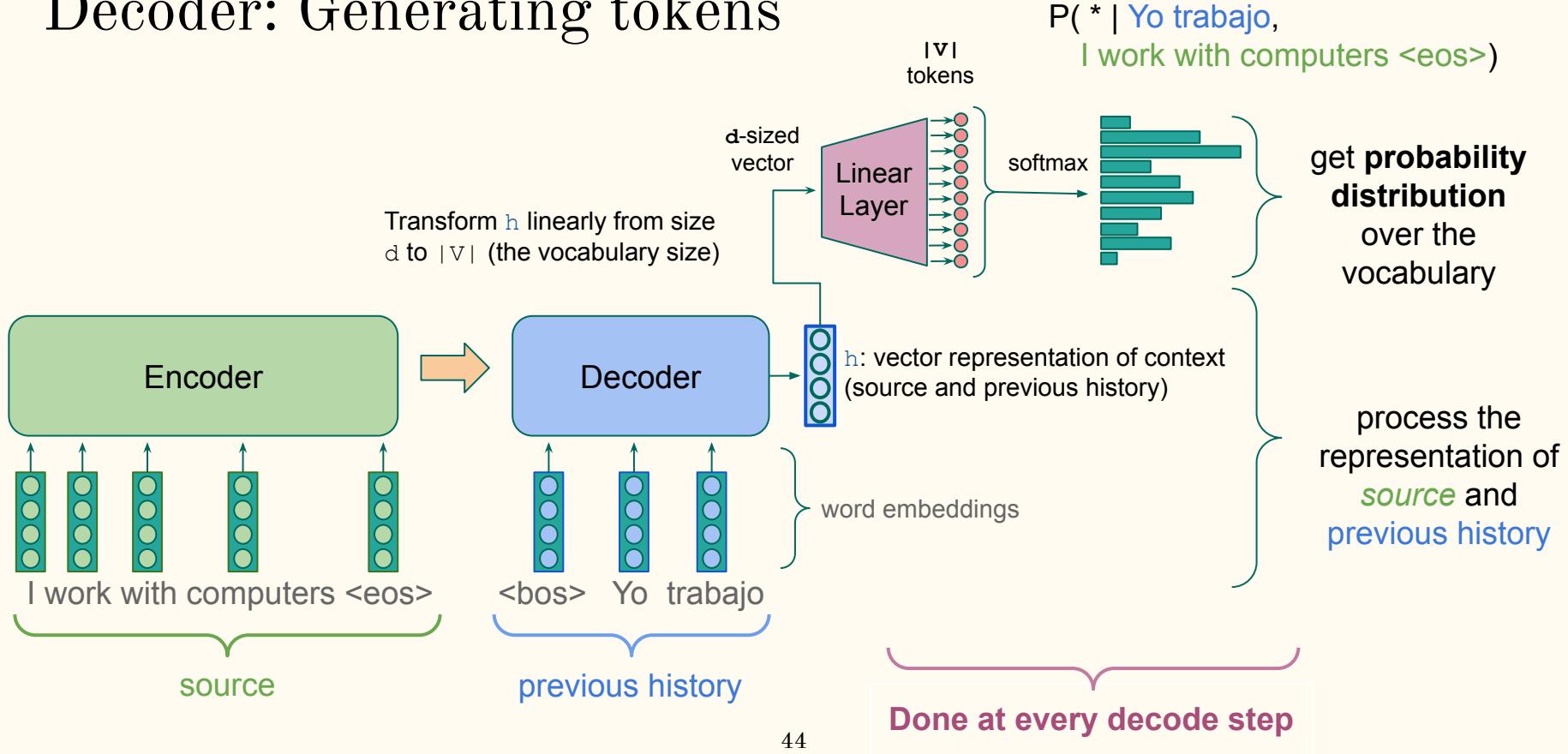
# Sequence Generation



# Decoder: Generating tokens

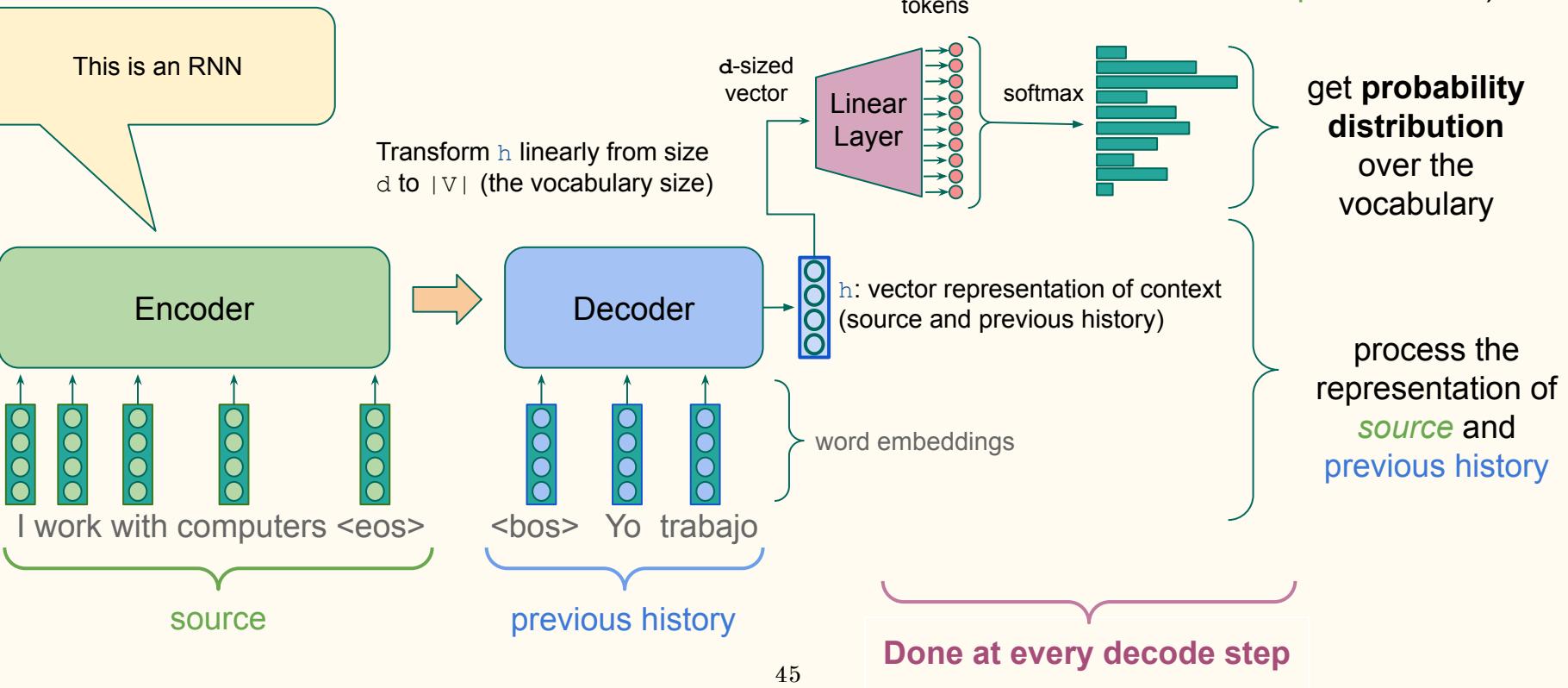


# Decoder: Generating tokens

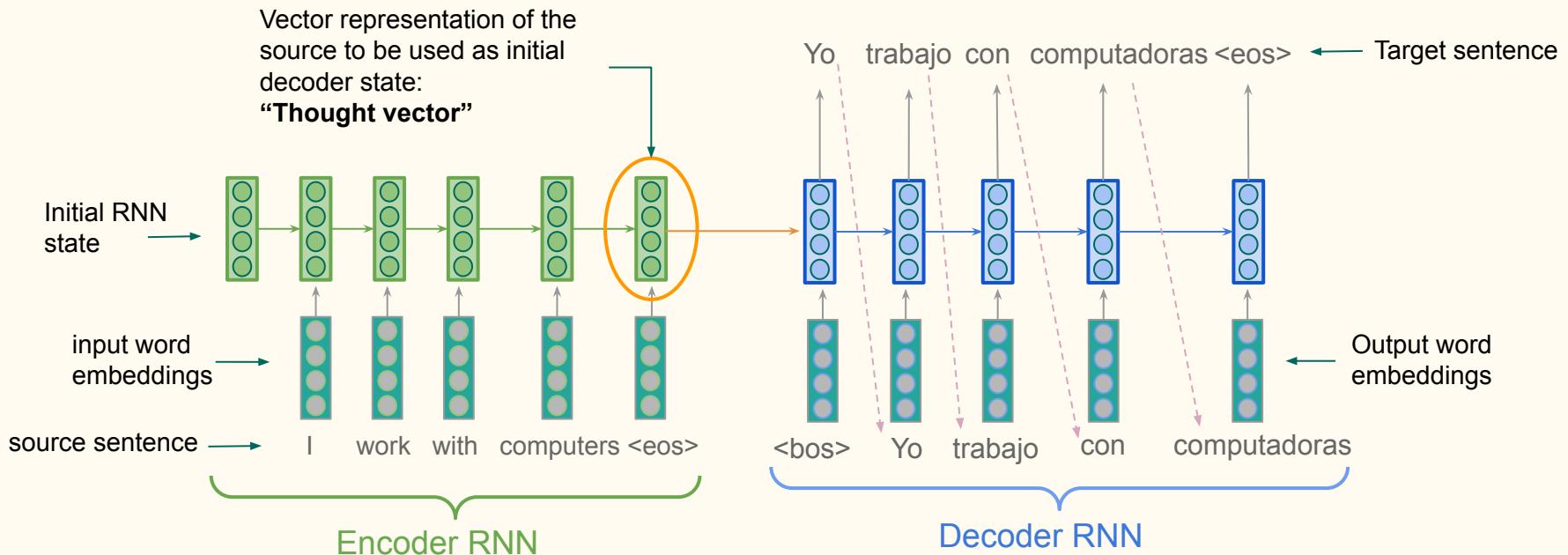


# Decoder: Generating tokens

This is an RNN



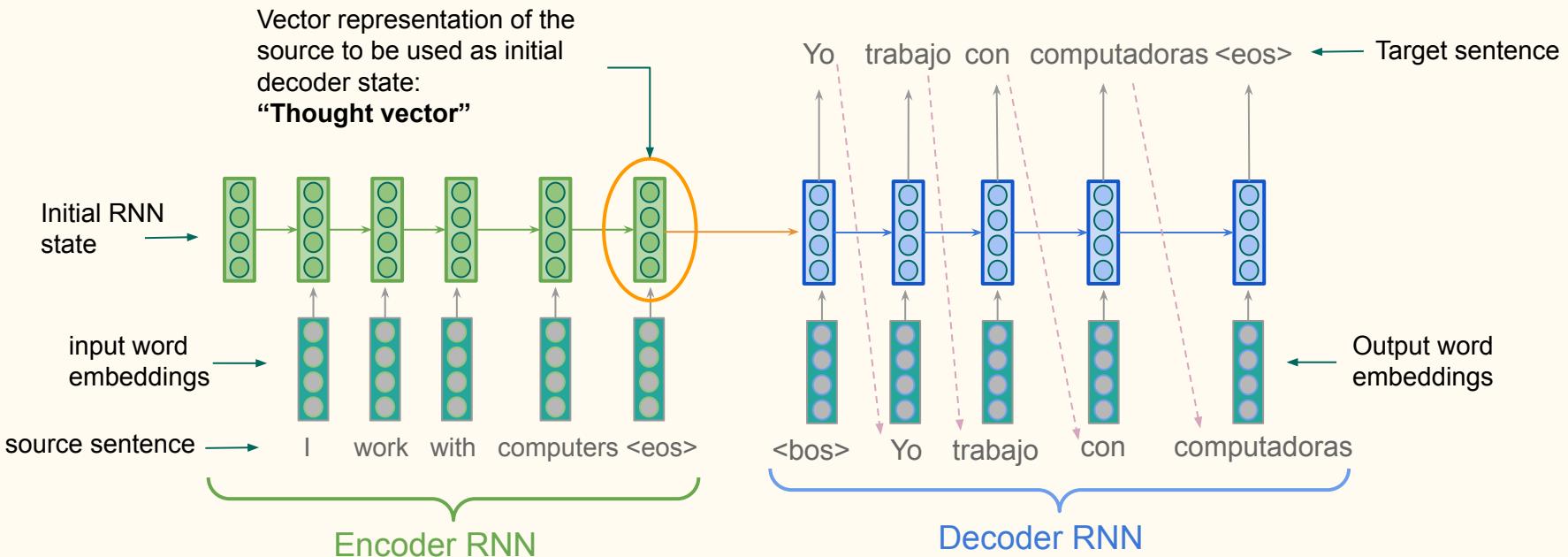
# Encoder-Decoder with RNNs



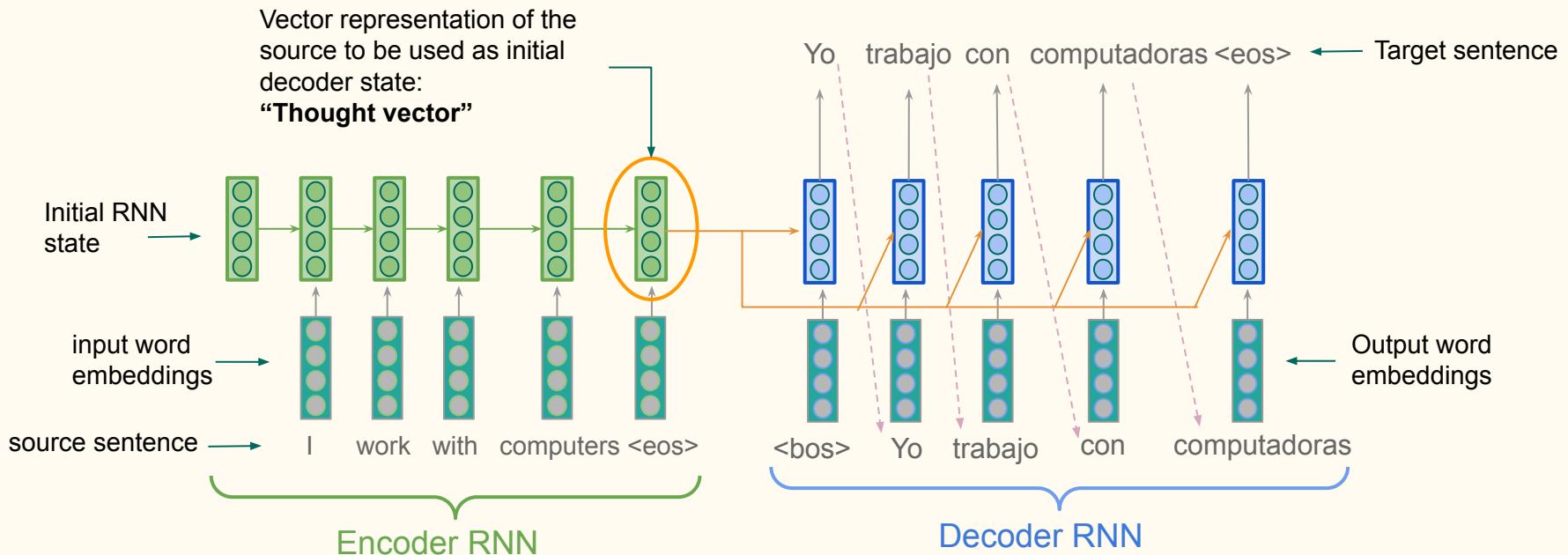
# Seq2Seq Architectures

- That this architecture works (after training) is surprising and is one of the more incredible NLP results.
- This is not the only possible architecture (although this has been shown to be the best)
- Also, in practice, these units are rarely RNNs. They are either GRU or LSTMs or BiLSTMs
- Models where information from previous time-steps is used to predict the output at the current time step are called **autoregressive**

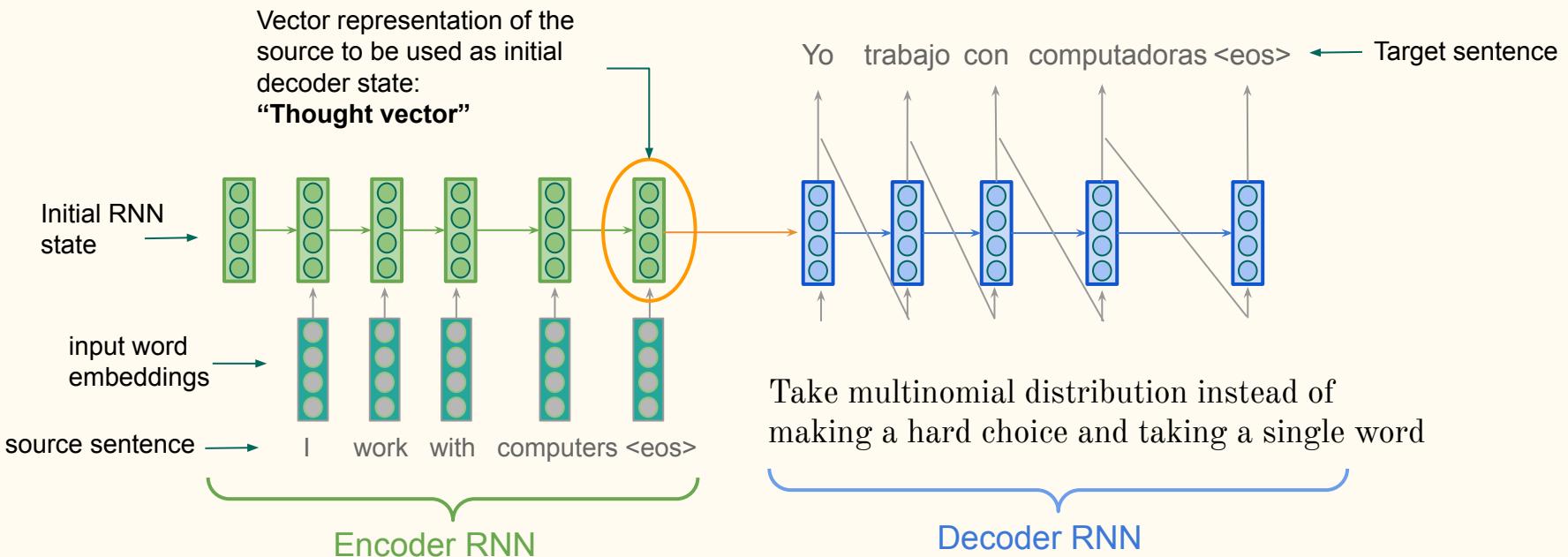
# *Can you think of alternatives?*



# Encoder-Decoder with RNNs: Alternative 1



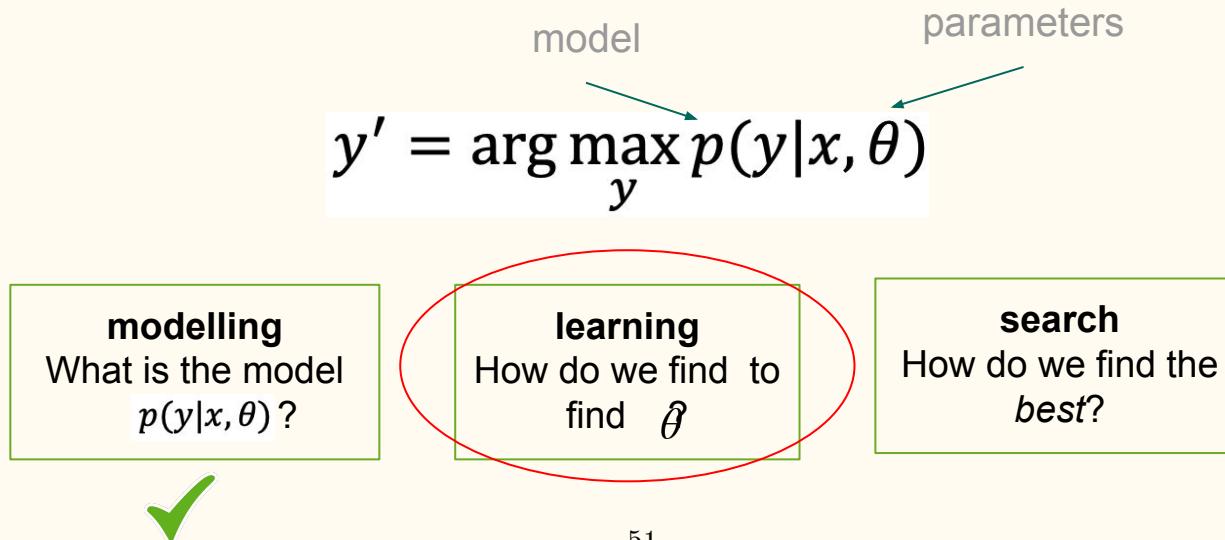
# Encoder-Decoder with RNNs: Alternative 2



# Task Formulation

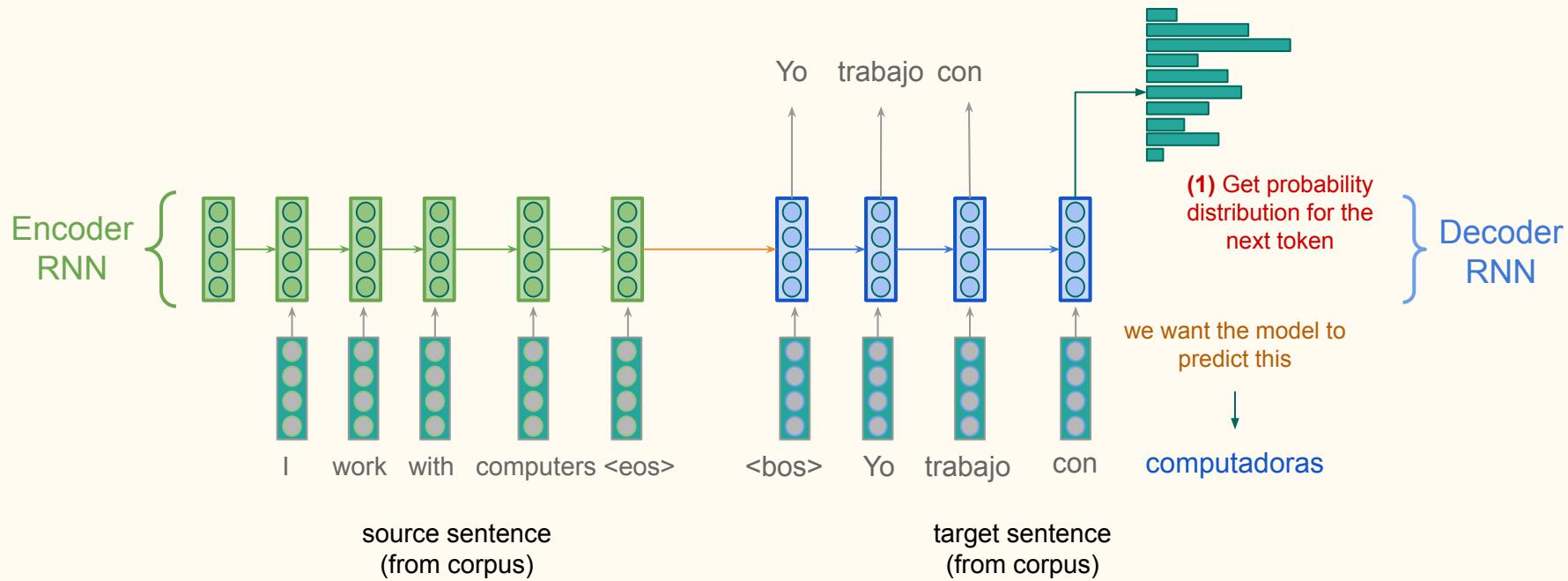
More Formally:

- We want to find best Spanish sentence  $y$ , given English sentence  $x$



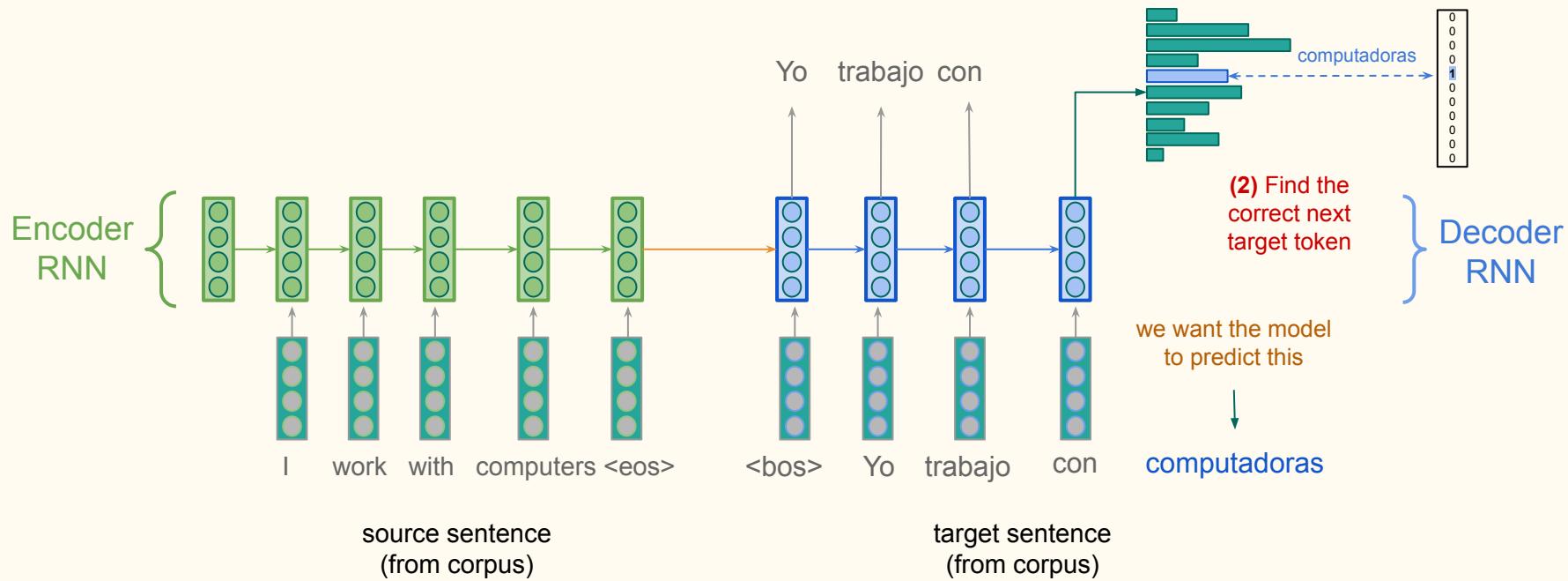
# Training with Cross-Entropy

$P( \cdot | \text{Yo trabajo con},$   
 $\text{I work with computers } \langle \text{eos} \rangle)$



# Training with Cross-Entropy

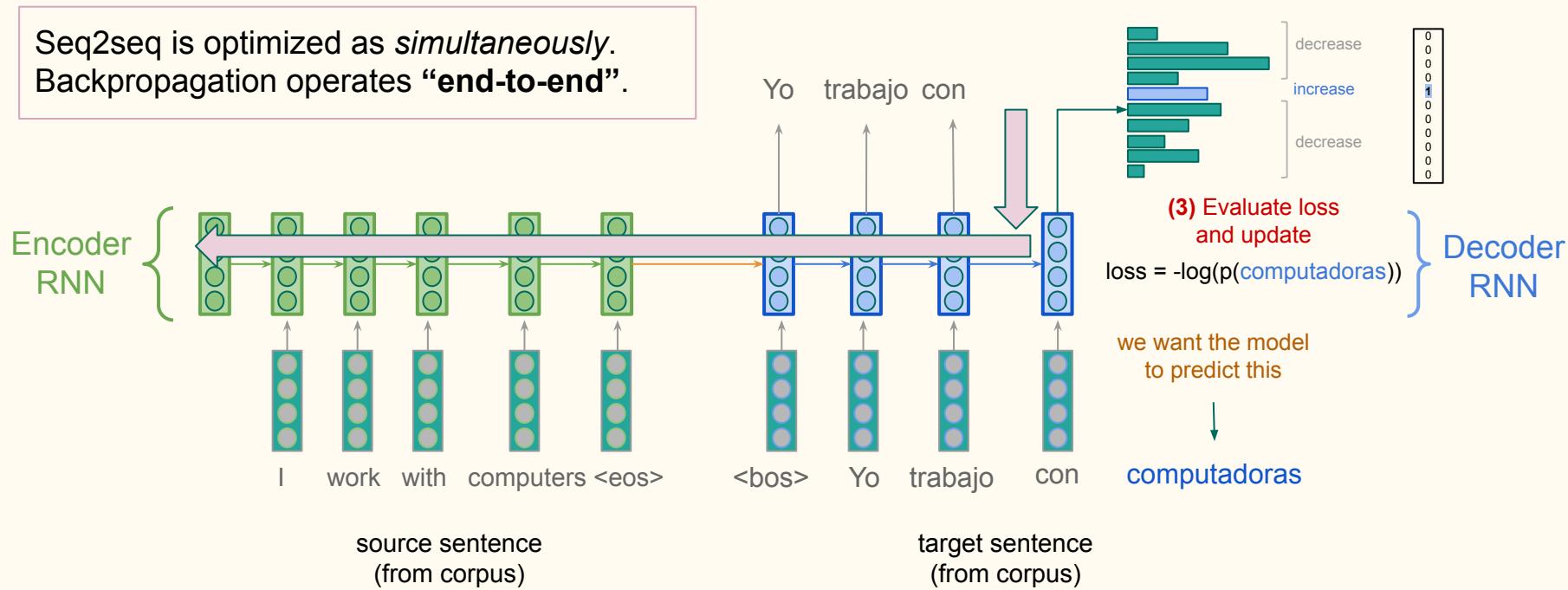
$P( * | \text{Yo trabajo con},$   
 $\text{I work with computers } \langle \text{eos} \rangle)$



# Training with Cross-Entropy

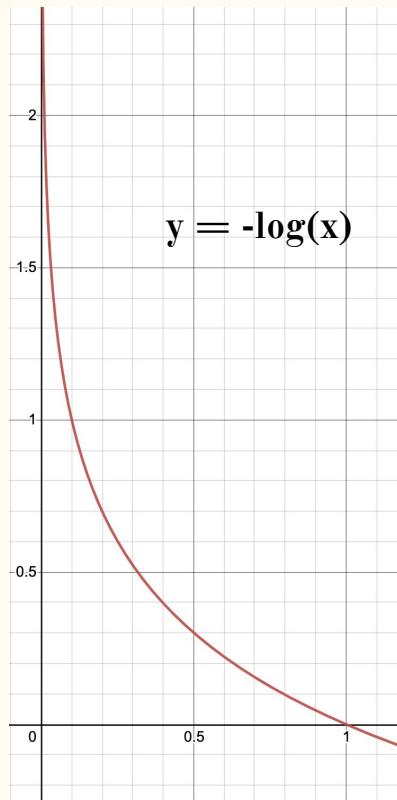
P( \* | Yo trabajo con,  
I work with computers <eos>)

Seq2seq is optimized as *simultaneously*.  
Backpropagation operates “**end-to-end**”.



# Recall: Cross-Entropy Loss

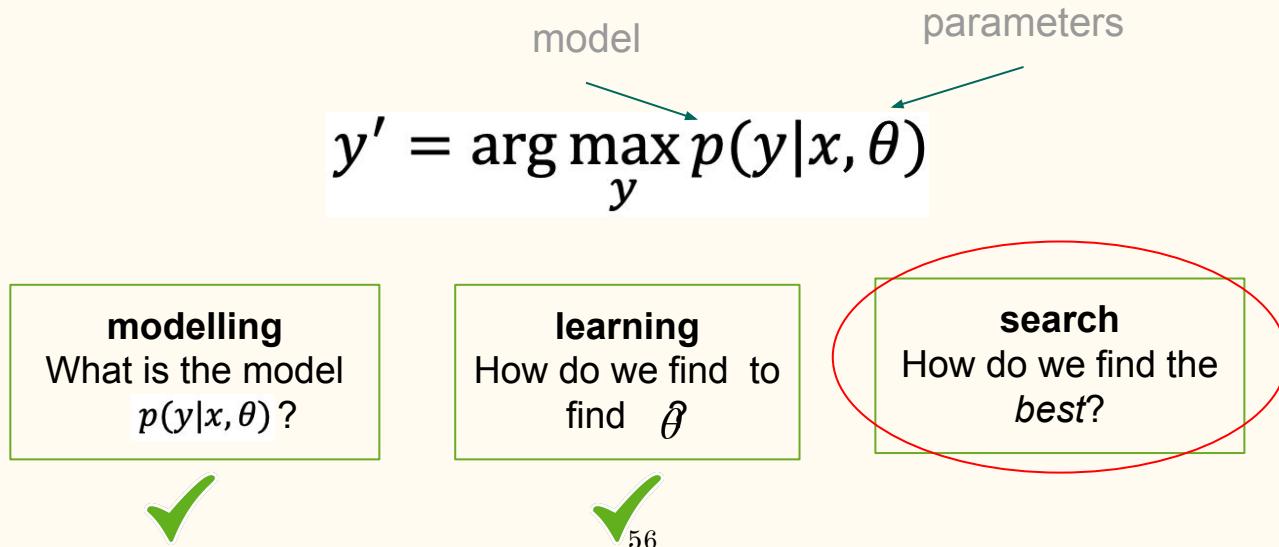
$$\text{loss} = -\log(p(\text{correct}))$$



# Task Formulation

More Formally:

- We want to find best Spanish sentence  $y$ , given English sentence  $x$



# Goal: Search for the best candidate

$$y' = \arg \max_y p(y|x) = \arg \max_y \prod_{t=1}^n p(y_t|y_{<t}, x)$$

How do we find  $\text{argmax}$ ?

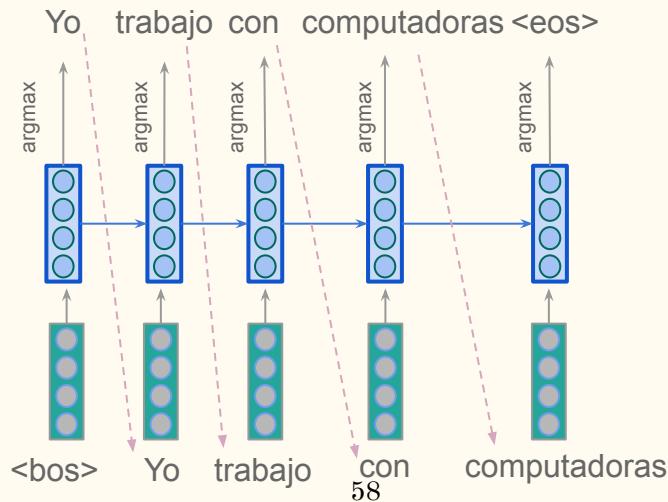
# A first attempt: Greedy Decoding

$$y' = \arg \max_y p(y|x) = \arg \max_y \prod_{t=1}^n p(y_t|y_{<t}, x)$$

How do we find argmax?

- **Greedy:** Take the argmax (most probable word) on each step of the decoder

Can't we generate all possible sequences?  
Too expensive!



***What is wrong with this solution?***

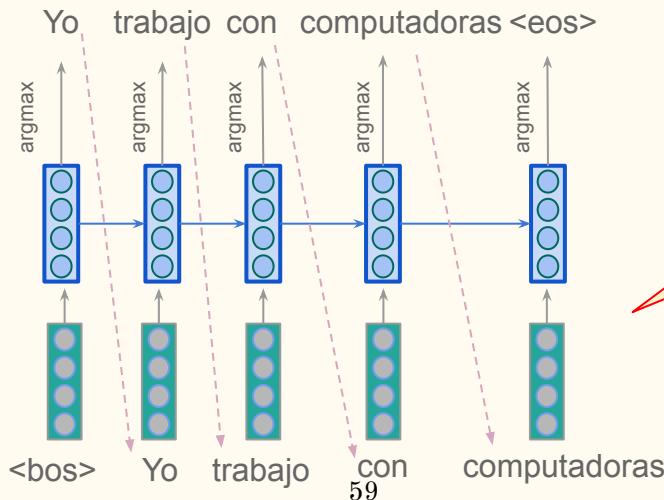
# A first attempt: Greedy Decoding

$$y' = \arg \max_y p(y|x) = \arg \max_y \prod_{t=1}^n p(y_t|y_{<t}, x)$$

How do we find argmax?

- **Greedy:** Take the argmax (most probable word) on each step of the decoder

Can't we generate all possible sequences?  
Too expensive!



**PROBLEM**  
The best token at the current step does not necessarily lead to the best sequence

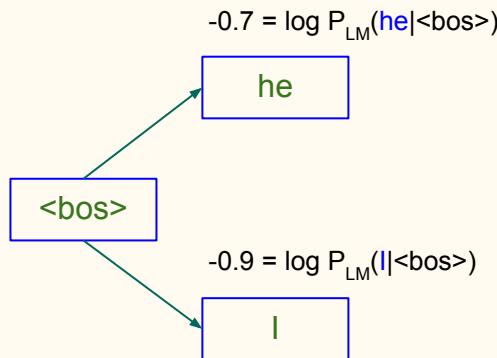
# Beam Search: Core Idea

- On each step of decoder, keep track of the **k** most probable partial translations (called **hypotheses**)
  - k is the **beam size** (in practice around 5 to 10)
- A hypothesis has a **score**:
$$\text{score}(y_1, \dots, y_t) = \log P_{\text{LM}}(y_1, \dots, y_t | x) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$
  - Scores are all negative, and **higher score is better**
  - We search for high-scoring hypotheses, tracking top k on each step
- Beam search is **not guaranteed** to find optimal solution
  - But it's efficient!

# Beam Search: Example

Beam Size =  $k = 2$

$$\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

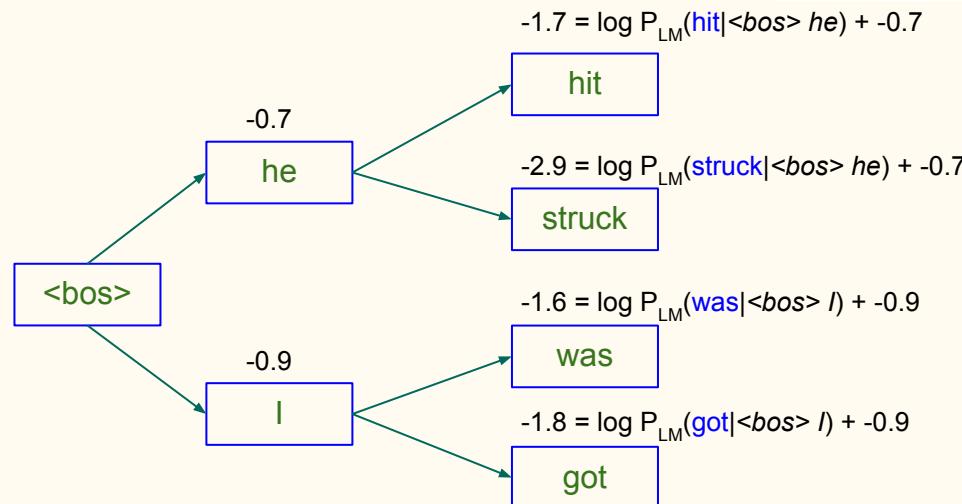


Find top  $k$  next words  
and calculate scores

# Beam Search: Example

Beam Size =  $k = 2$

$$\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

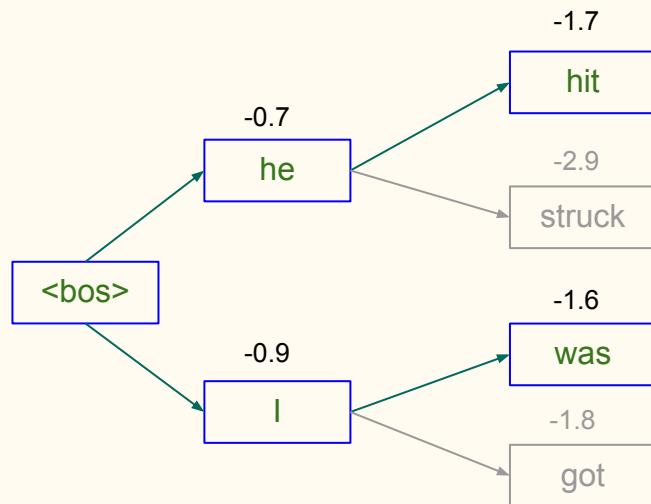


For each of the  $k$  hypotheses, find top  $k$  next words and calculate scores

# Beam Search: Example

Beam Size =  $k = 2$

$$\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

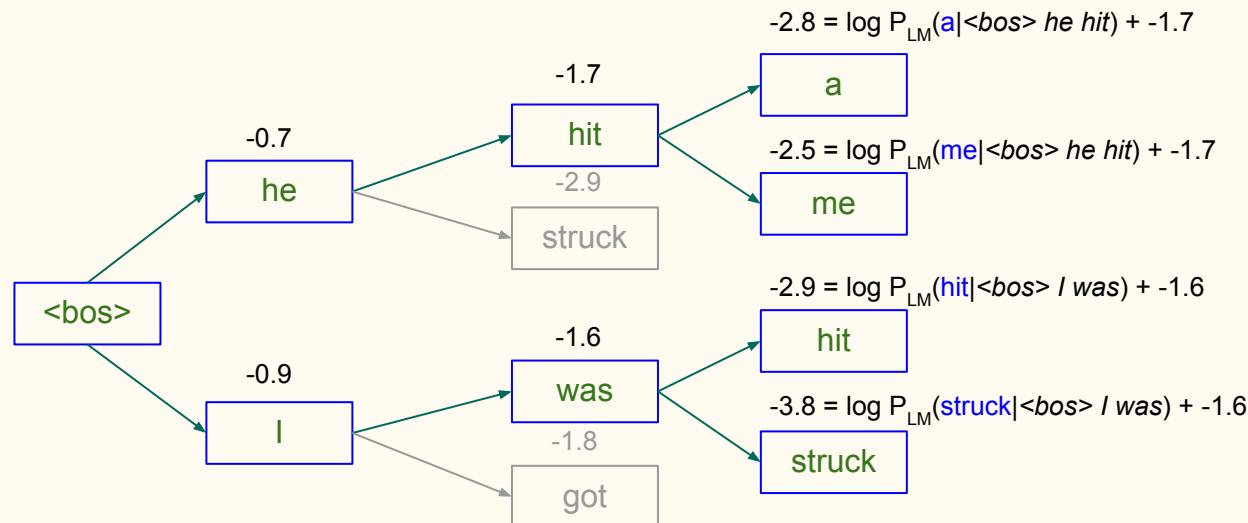


Of these  $k^2$  hypotheses,  
just keep  $k$  with highest scores

# Beam Search: Example

Beam Size =  $k = 2$

$$\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

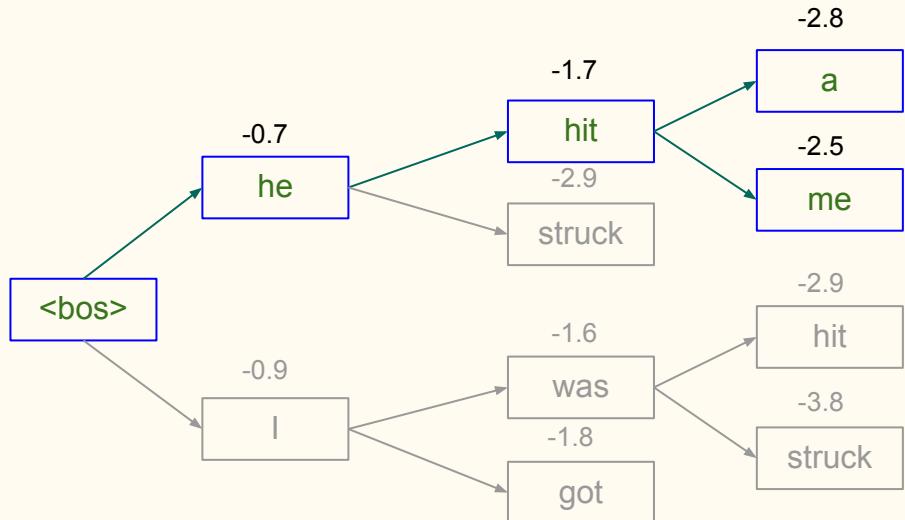


For each of the  $k$  hypotheses, find top  $k$  next words and calculate scores

# Beam Search: Example

Beam Size =  $k = 2$

$$\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

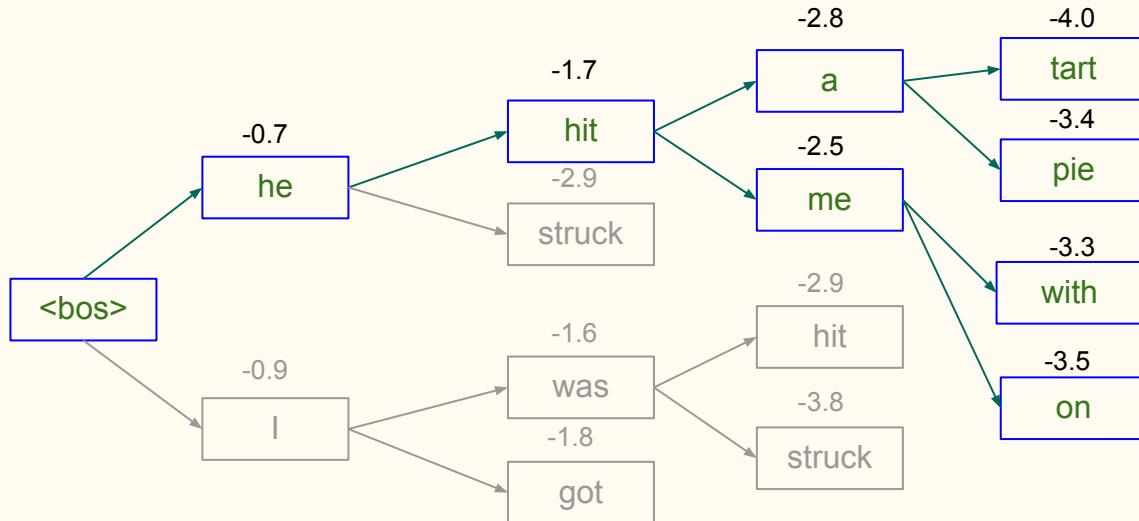


Of these  $k^2$  hypotheses,  
just keep  $k$  with highest scores

# Beam Search: Example

Beam Size =  $k = 2$

$$\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

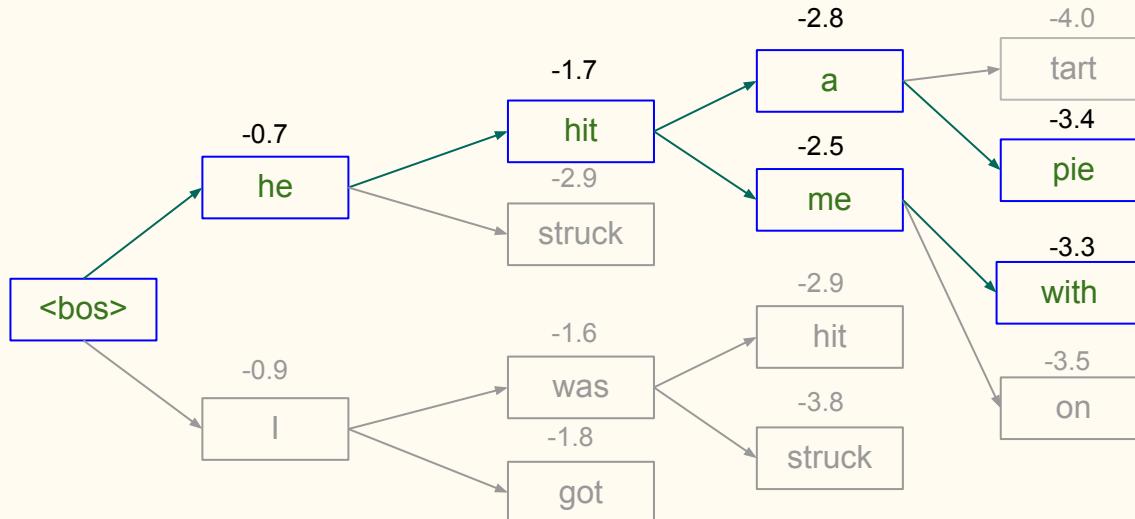


For each of the  $k$  hypotheses, find  
top  $k$  next words and calculate scores

# Beam Search: Example

Beam Size =  $k = 2$

$$\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

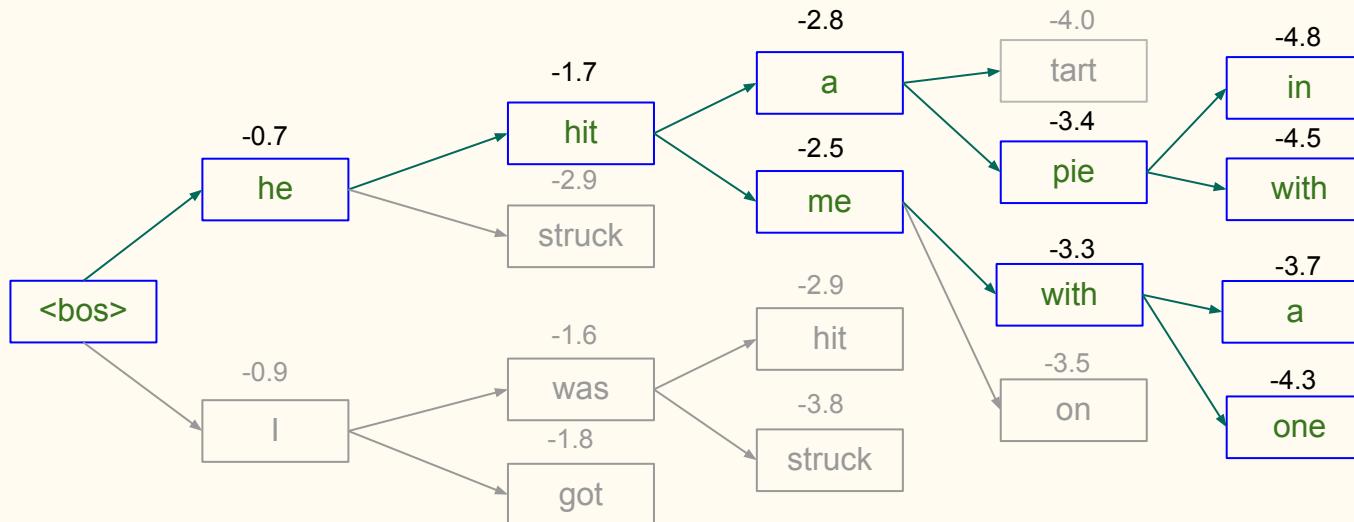


Of these  $k^2$  hypotheses,  
just keep  $k$  with highest scores

# Beam Search: Example

Beam Size =  $k = 2$

$$\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

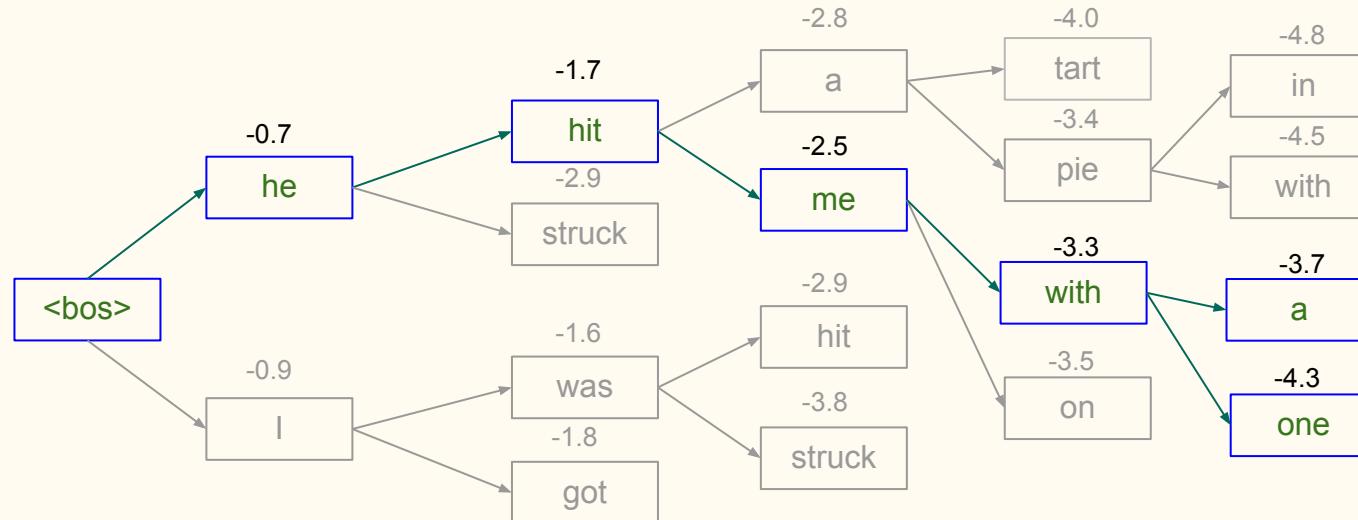


For each of the  $k$  hypotheses, find top  $k$  next words and calculate scores

# Beam Search: Example

Beam Size =  $k = 2$

$$\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

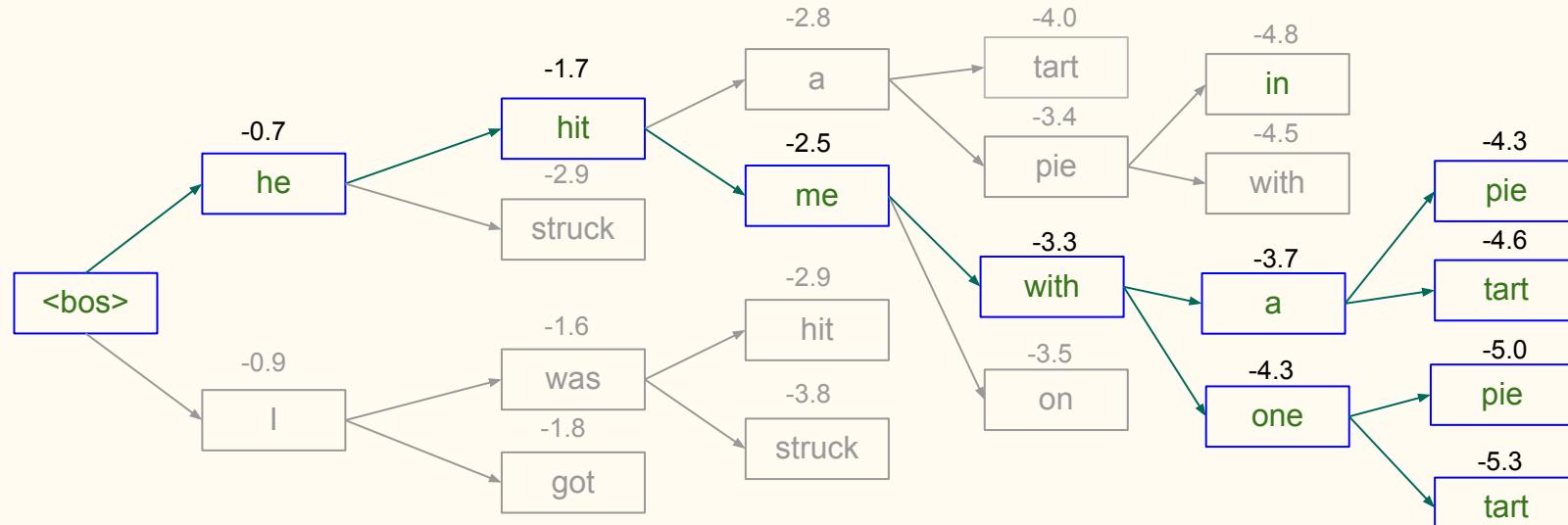


Of these  $k^2$  hypotheses,  
just keep  $k$  with highest scores

# Beam Search: Example

Beam Size =  $k = 2$

$$\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

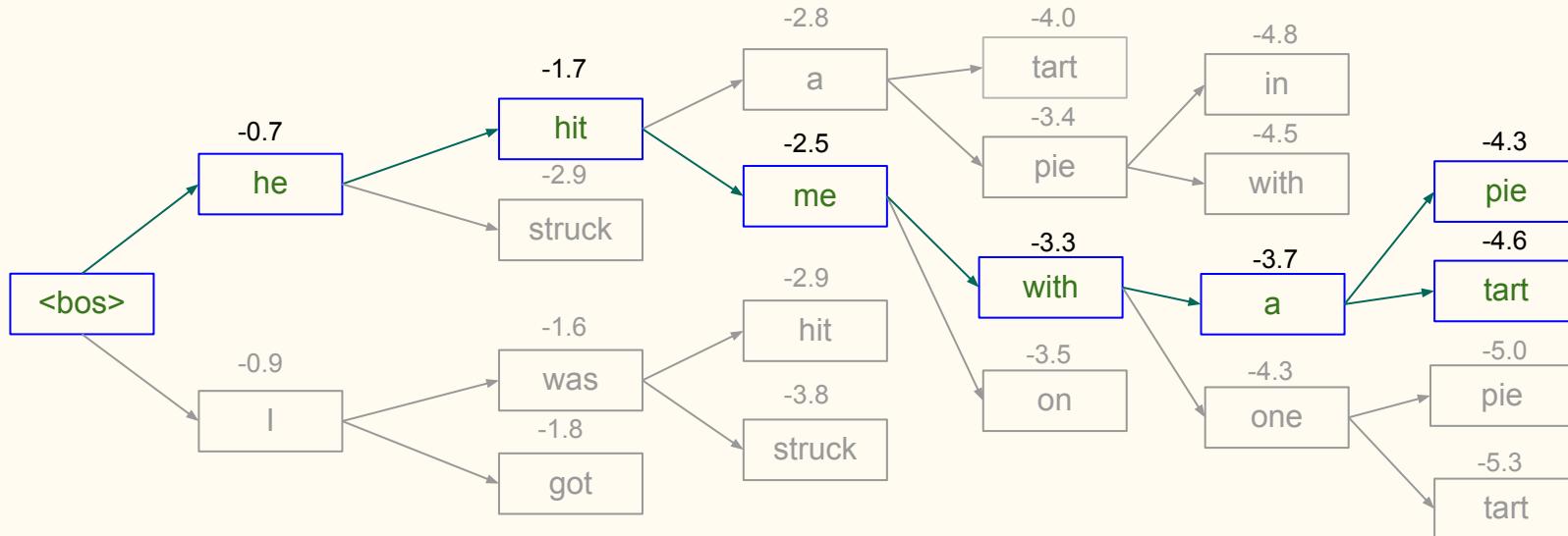


For each of the  $k$  hypotheses, find top  $k$  next words and calculate scores

# Beam Search: Example

Beam Size =  $k = 2$

$$\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

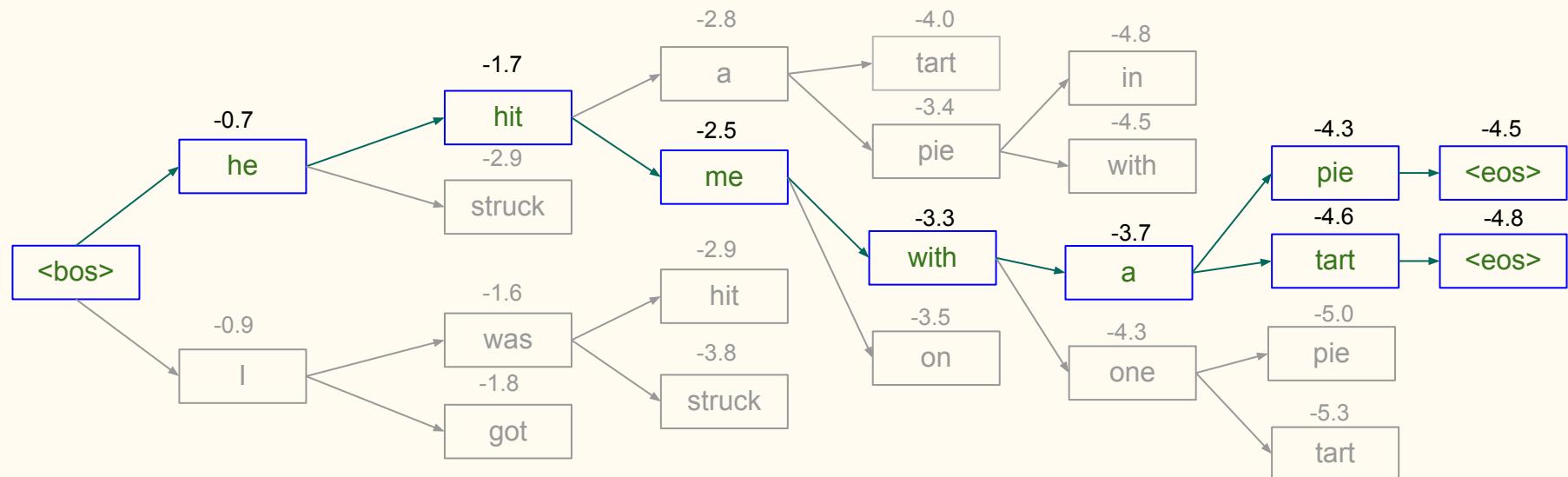


Of these  $k^2$  hypotheses,  
just keep  $k$  with highest scores

# Beam Search: Example

Beam Size =  $k = 2$

$$\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

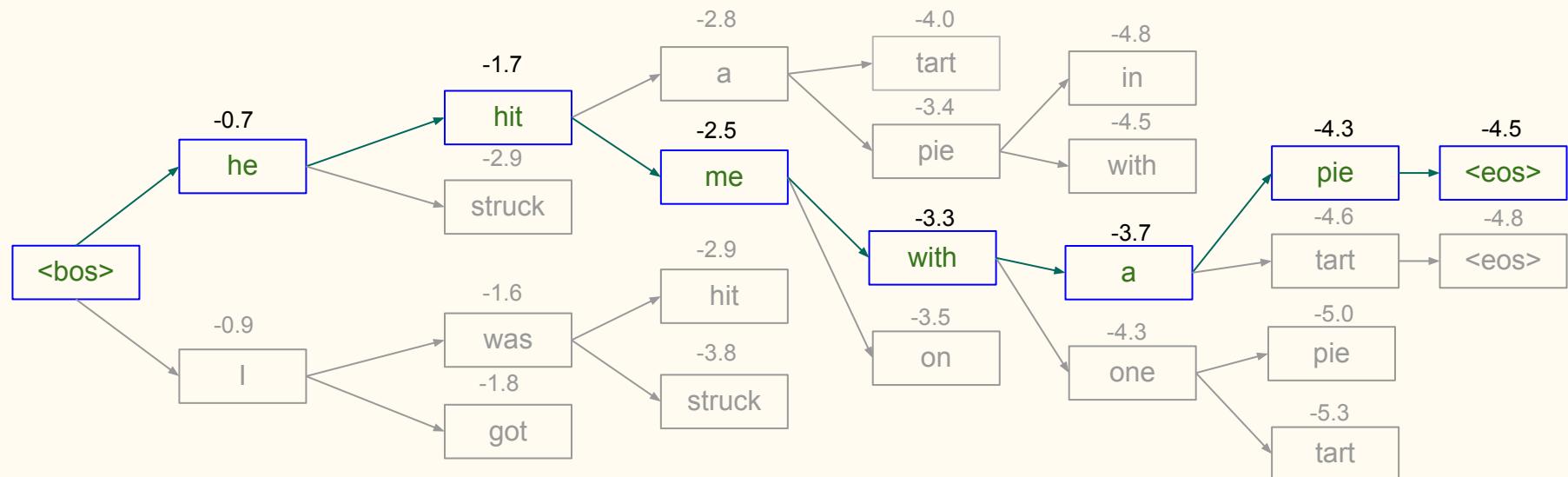


For each of the  $k$  hypotheses, find top  $k$  next words and calculate scores

# Beam Search: Example

Beam Size =  $k = 2$

$$\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$



We found the top-scoring hypothesis!

# Beam Search: Stopping Criteria

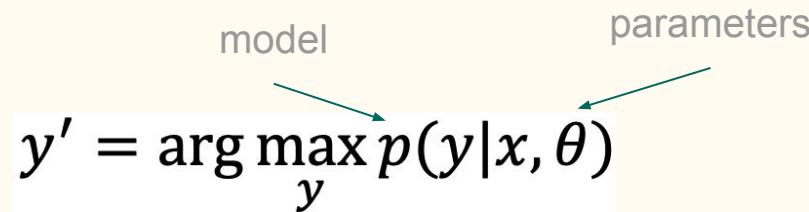
- Different hypotheses may produce `<eos>` tokens on different timesteps
  - When a hypothesis produces `<eos>`, that hypothesis is complete
  - Place it aside and continue exploring other hypotheses via beam search
- Usually we continue beam search until:
  - We reach timestep  $T$  (where  $T$  is some pre-defined cutoff), or
  - We have at least  $n$  completed hypotheses (where  $n$  is pre-defined cutoff)

# Task Formulation

- Suppose we are translating French → English
- We want to find best English sentence  $y$ , given French sentence  $x$

$$y' = \arg \max_y p(y|x, \theta)$$

model                                    parameters



**modelling**  
How's the model  
for  $p(y|x, \theta)$  ?



**learning**  
How to find  $\theta$  ?

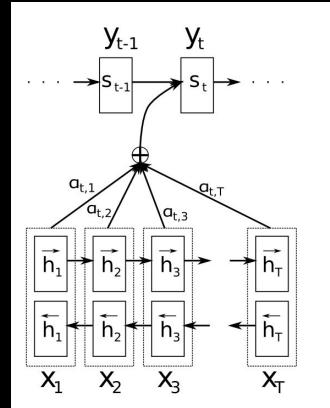


**search**  
How to find the  
argmax?



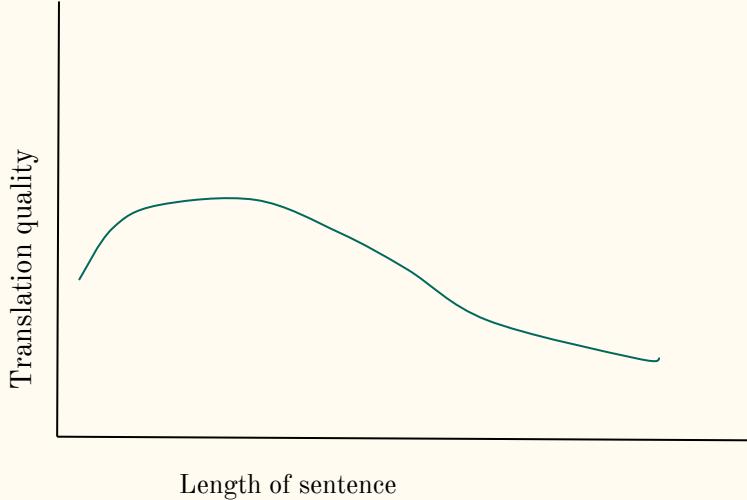
# In this Section...

- Seq2Seq models are implemented with an **Encoder-Decoder** architecture
  - The Encoder creates a representation of the **source** sentence (thought vector)
  - The Decoder uses that representation to generate the **target** sentence
- Encoder and Decoder can be implemented using RNNs
  - These are trained end-to-end using **cross-entropy loss**
  - For efficient inference (test time), we can use **beam search**



# Improvement Attention

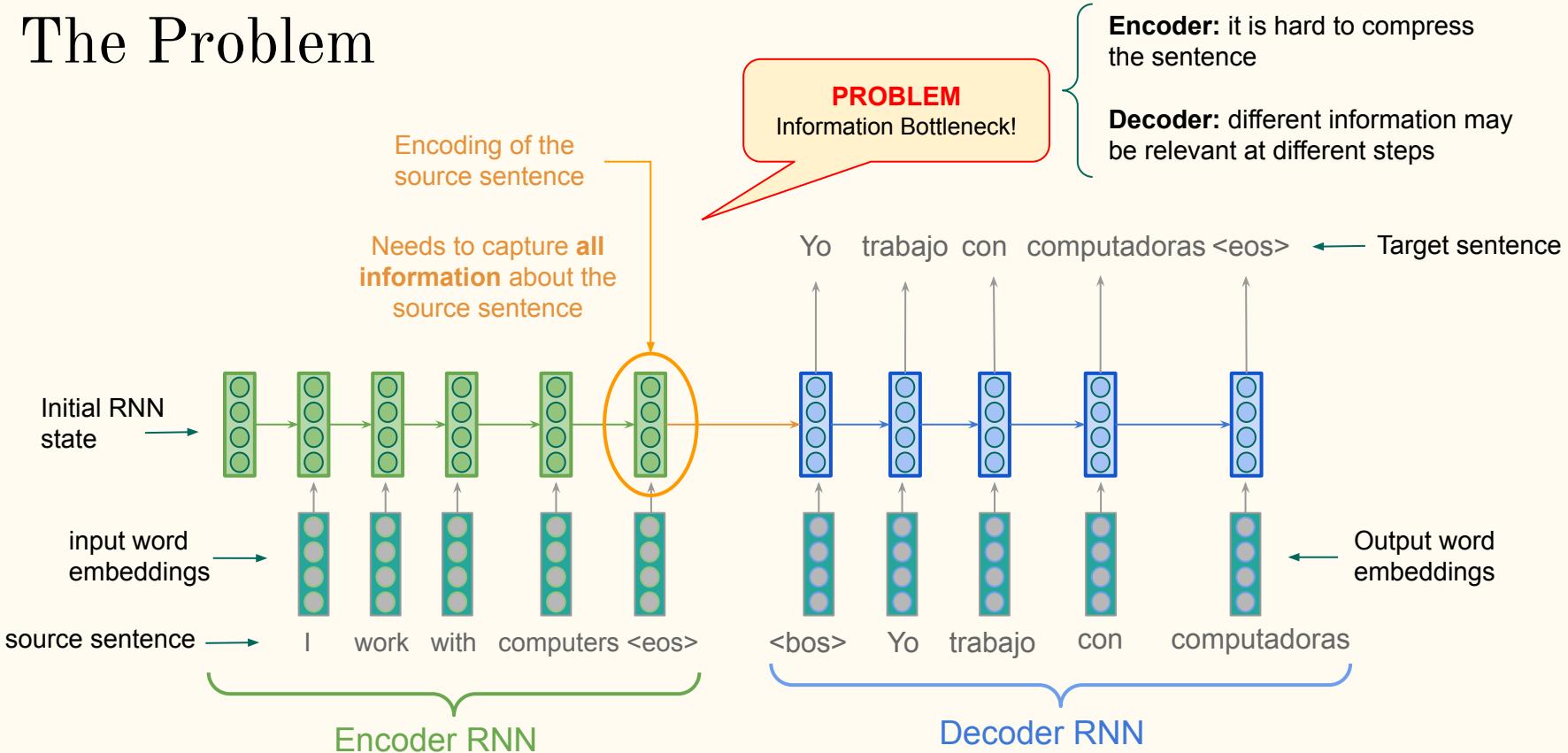
# The Problem



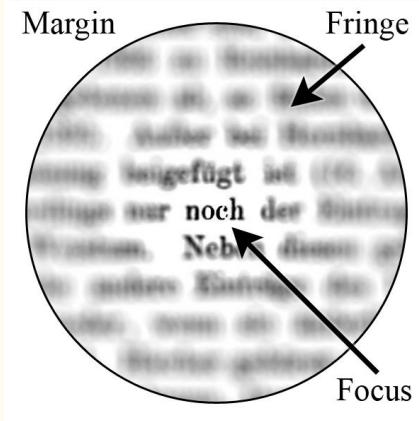
Translation quality drops with increase in the length of sentences.

(Notice that the encoder has to “pack” more information into the same vector representation or thought vector)

# The Problem



# Attention: A Biological Motivation



Focusing on specific information is the only way to avoid information overload.

Critical to biological systems. (Only way to avoid information overload, remember information processing is expensive)

# Attention

We need a way to allow models to **attend to** (focus on) different parts of the input at different timesteps. (Attention was previously used in vision  
Larochelle & Hinton, 2010)

## NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

**Dzmitry Bahdanau**

Jacobs University Bremen, Germany

**KyungHyun Cho      Yoshua Bengio\***

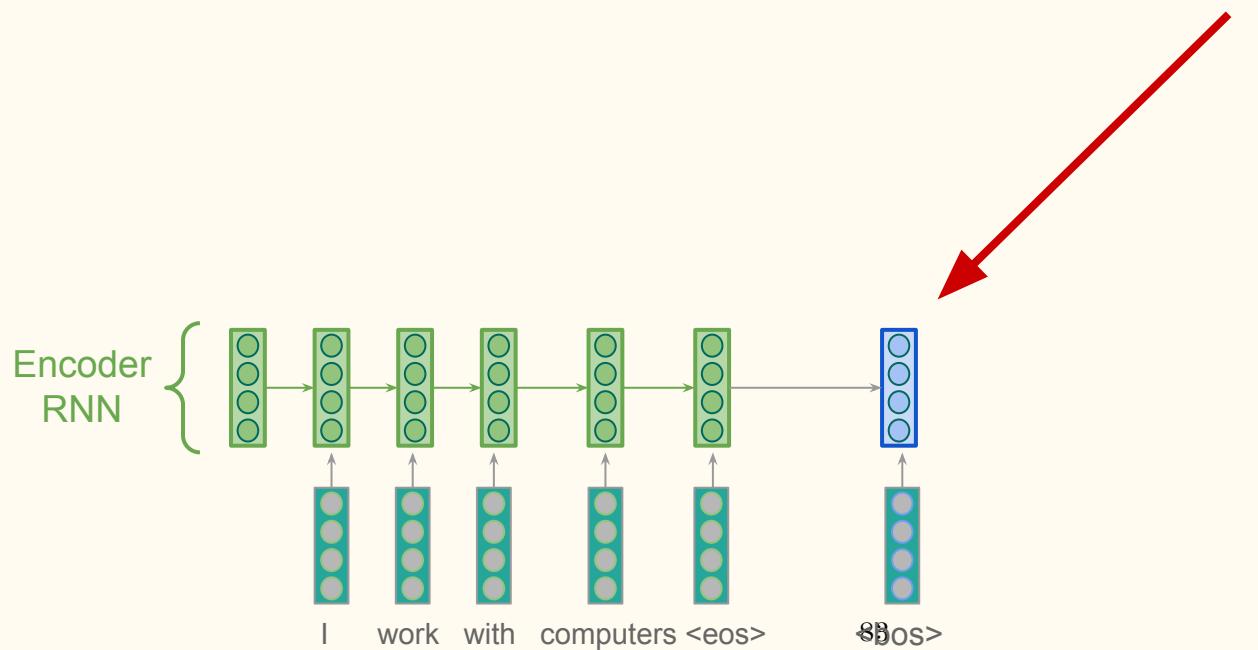
Université de Montréal

# Attention

- On each step of the decoder, use **direct connection to the encoder** to **focus on a particular part** of the source sequence
  - Introduced in “*Neural Machine Translation by Jointly Learning to Align and Translate*” (Bahdanau et al., 2015)
- The encoder does not have to compress the whole source into a single vector
  - It gives **representations for all source tokens** (e.g., all RNN states instead of the last one)

# Sequence-to-Sequence with Attention: The Goal

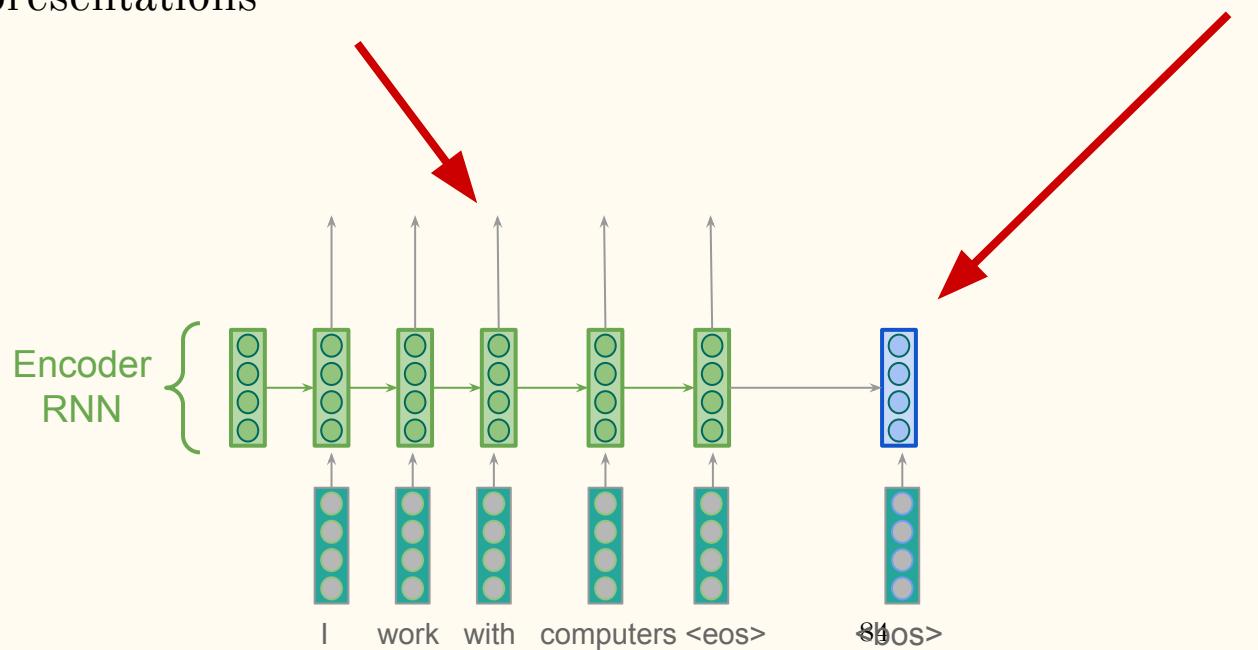
In generating the output at this step



# Sequence-to-Sequence with Attention: The Goal

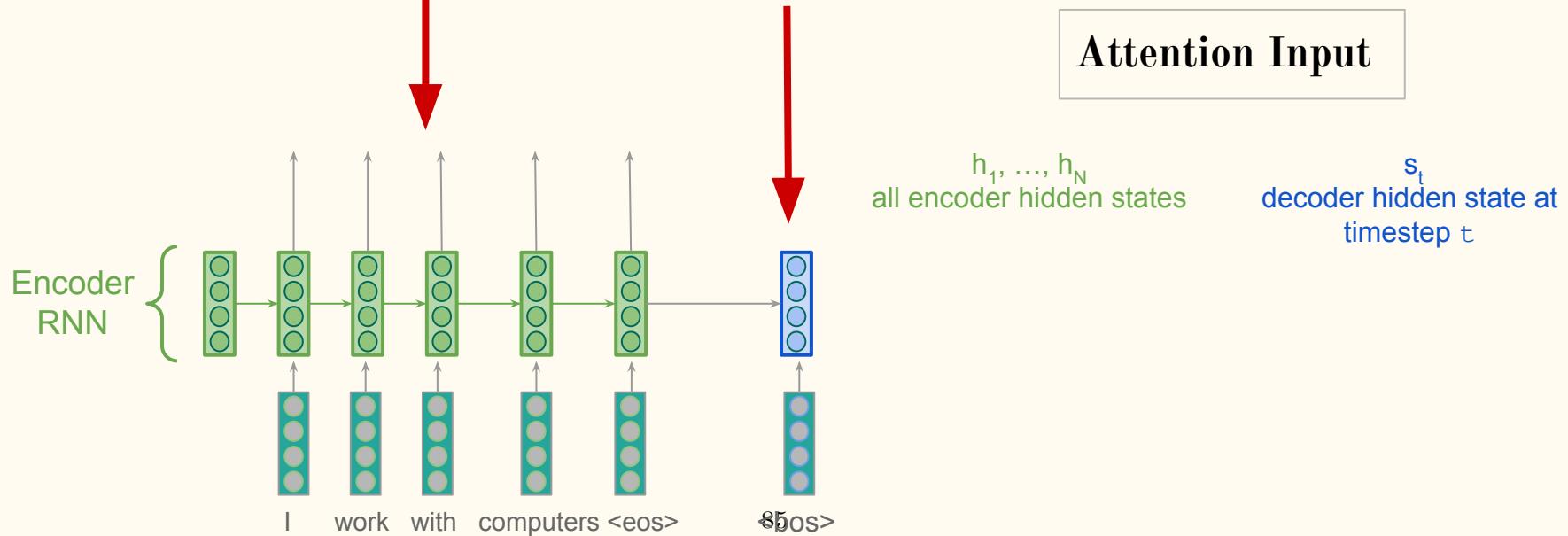
We want to make use of some of these representations

In generating the output at this step



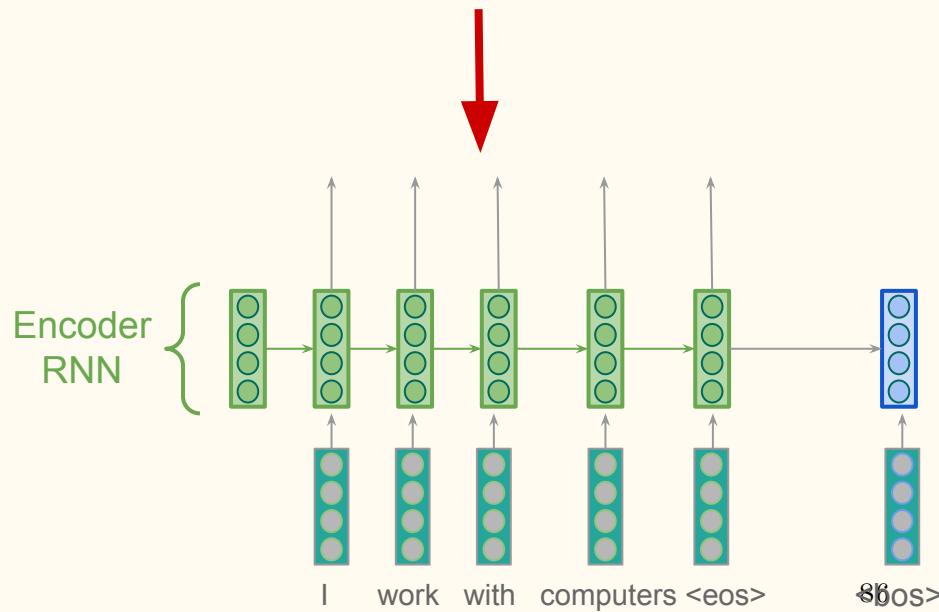
# Sequence-to-Sequence with Attention

We use a weighted sum of these based on how “important” each is to generate this output



# Sequence-to-Sequence with Attention

Step 1: Find out how “important”  
each of these are.



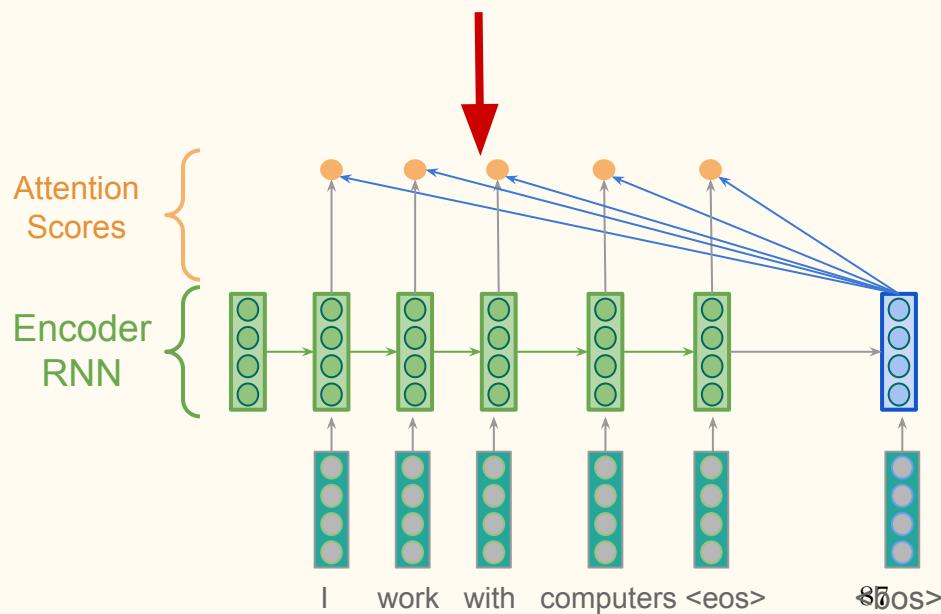
# Sequence-to-Sequence with Attention

Step 1: Find out how “important”  
each of these are.

## Attention Scores

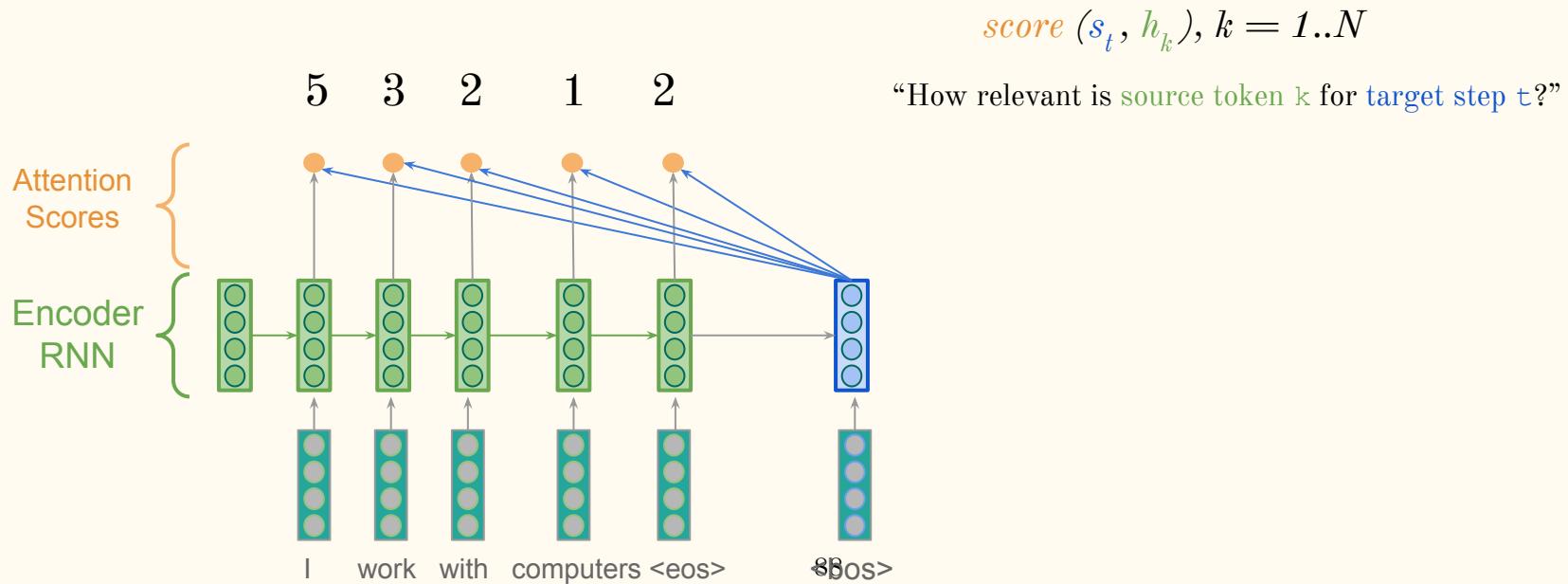
$$\text{score}(s_t, h_k), k = 1..N$$

“How relevant is source token  $k$  for target step  $t$ ? ”

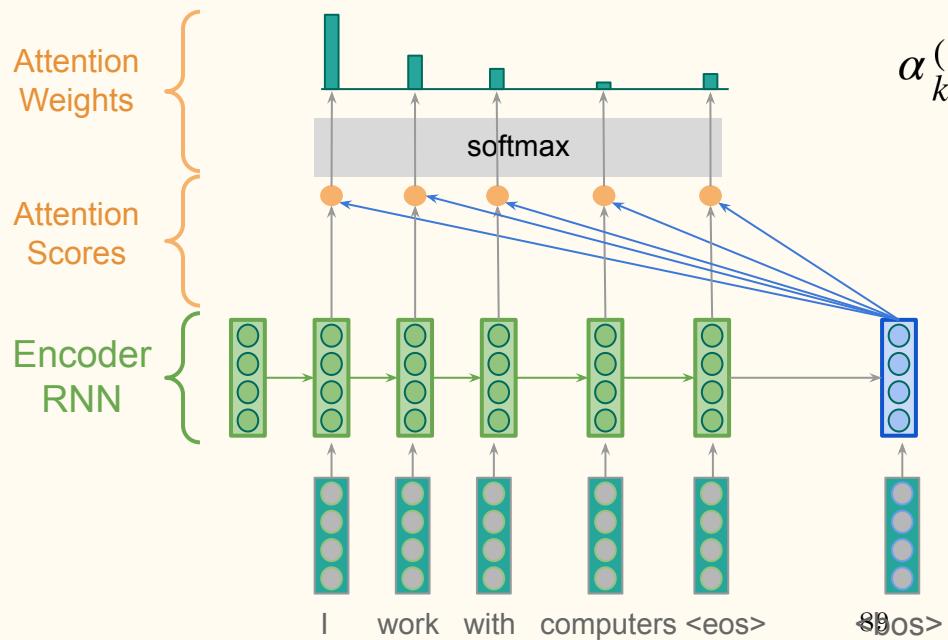


# Sequence-to-Sequence with Attention

## Attention Scores



# Sequence-to-Sequence with Attention

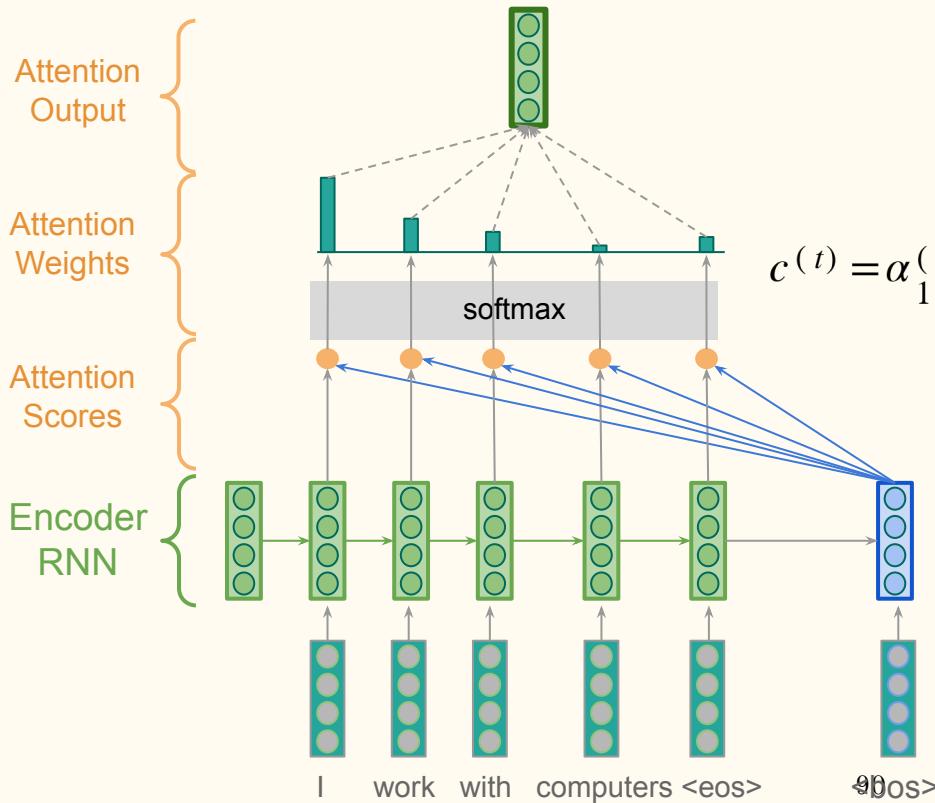


## Attention Weights

$$\alpha_k^{(t)} = \frac{\exp(\text{score}(s_t, h_k))}{\sum_{i=1}^N \exp(\text{score}(s_t, h_i))}, \quad k = 1..N$$

**WARNING:** Do not confuse attention weights with learnable weights.

# Sequence-to-Sequence with Attention

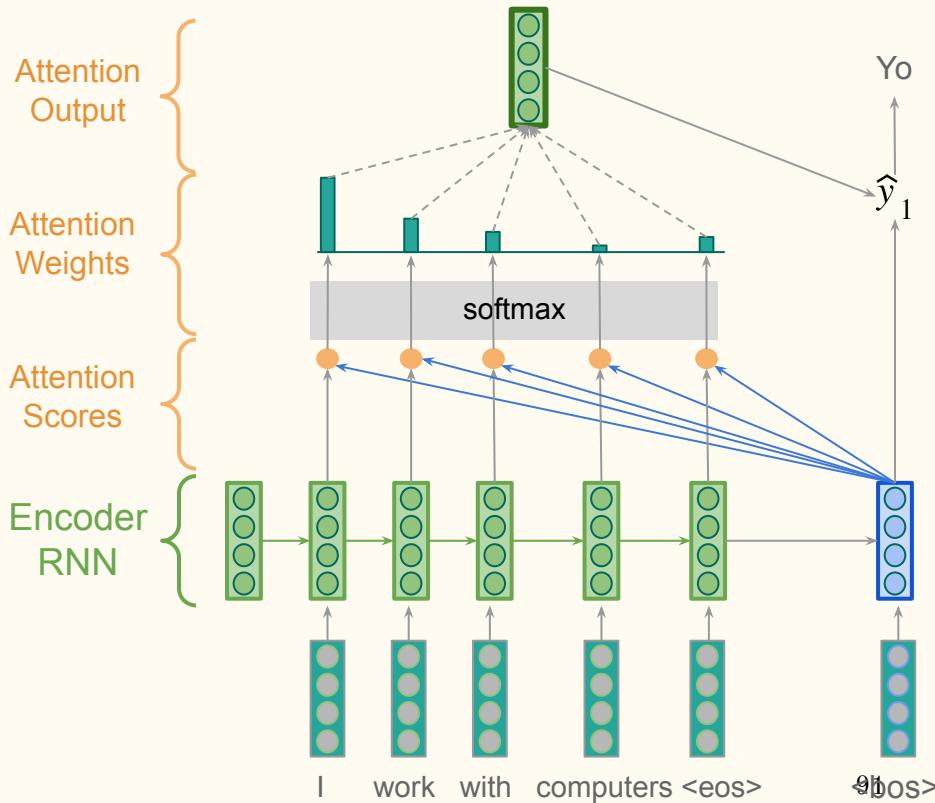


Attention output

$$c^{(t)} = \alpha_1^{(t)} h_1 + \alpha_2^{(t)} h_2 + \dots + \alpha_N^{(t)} h_N = \sum_{k=1}^N \alpha_k^{(t)} h_k$$

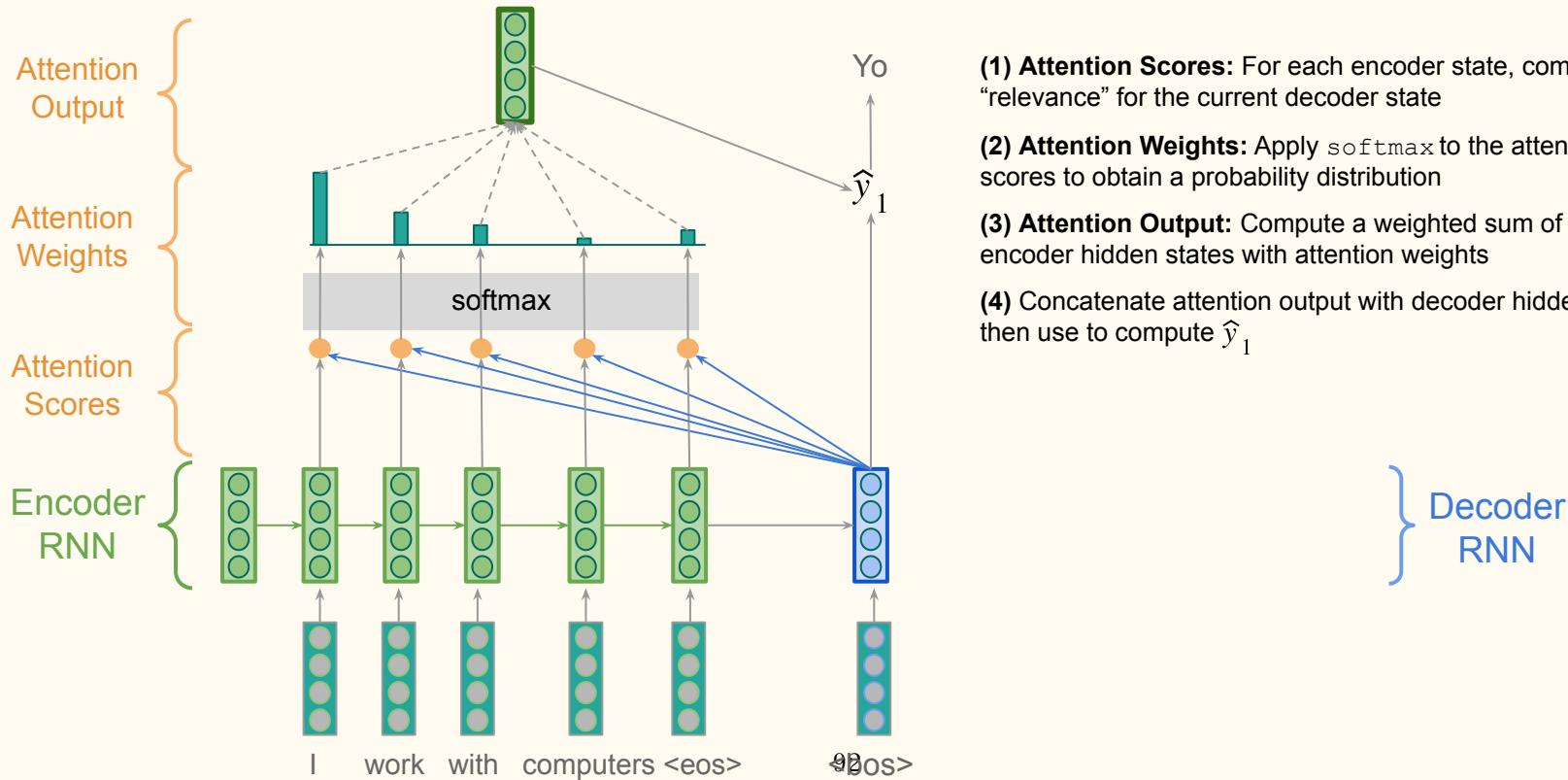
source context for decoder step  $t$

# Sequence-to-Sequence with Attention



Now combine the **attention output** with the **internal representation** of the decoder

# Sequence-to-Sequence with Attention



**(1) Attention Scores:** For each encoder state, compute its “relevance” for the current decoder state

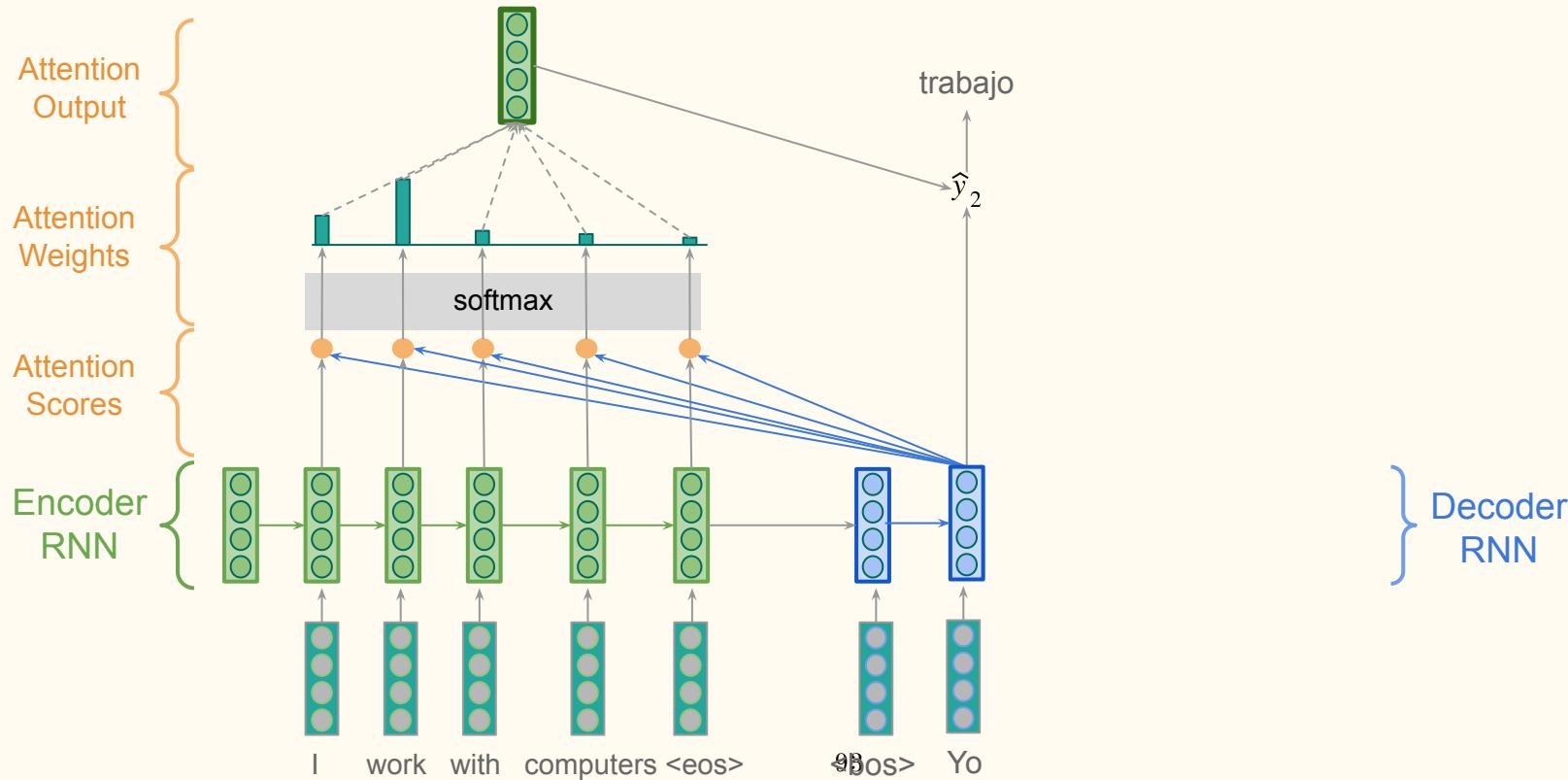
**(2) Attention Weights:** Apply softmax to the attention scores to obtain a probability distribution

**(3) Attention Output:** Compute a weighted sum of the encoder hidden states with attention weights

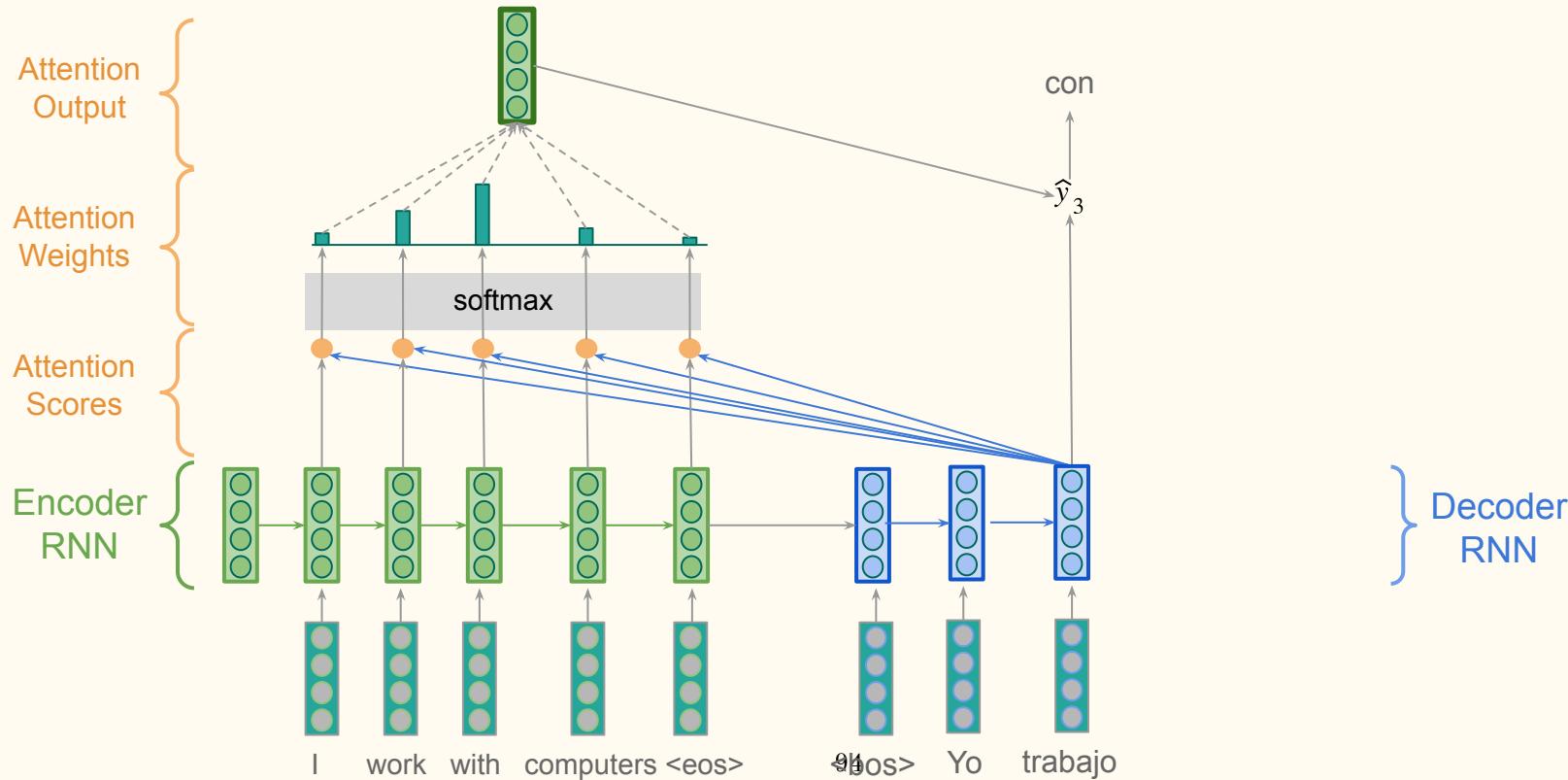
**(4) Concatenate attention output with decoder hidden state, then use to compute  $\hat{y}_1$**

} Decoder RNN

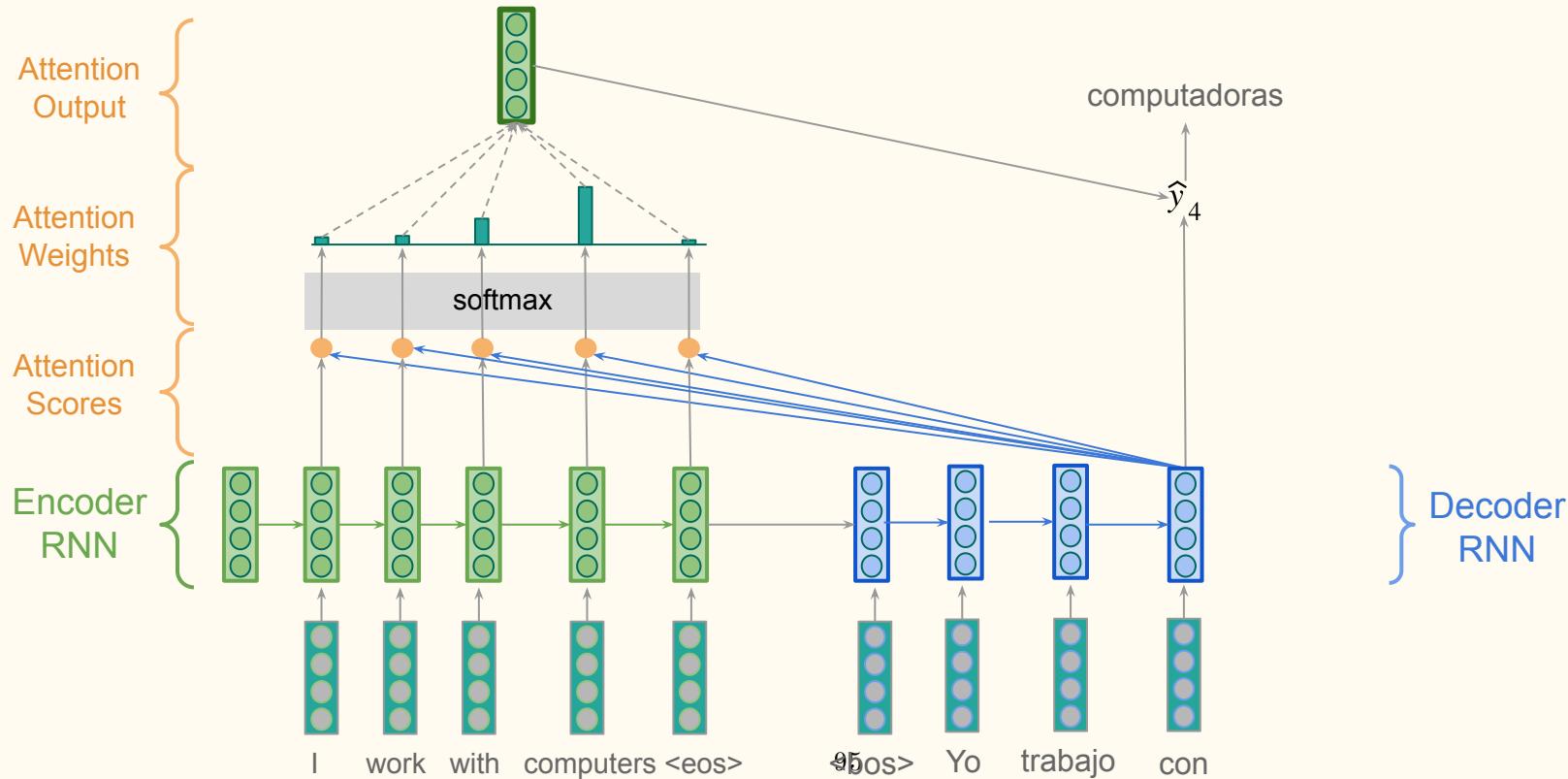
# Sequence-to-Sequence with Attention



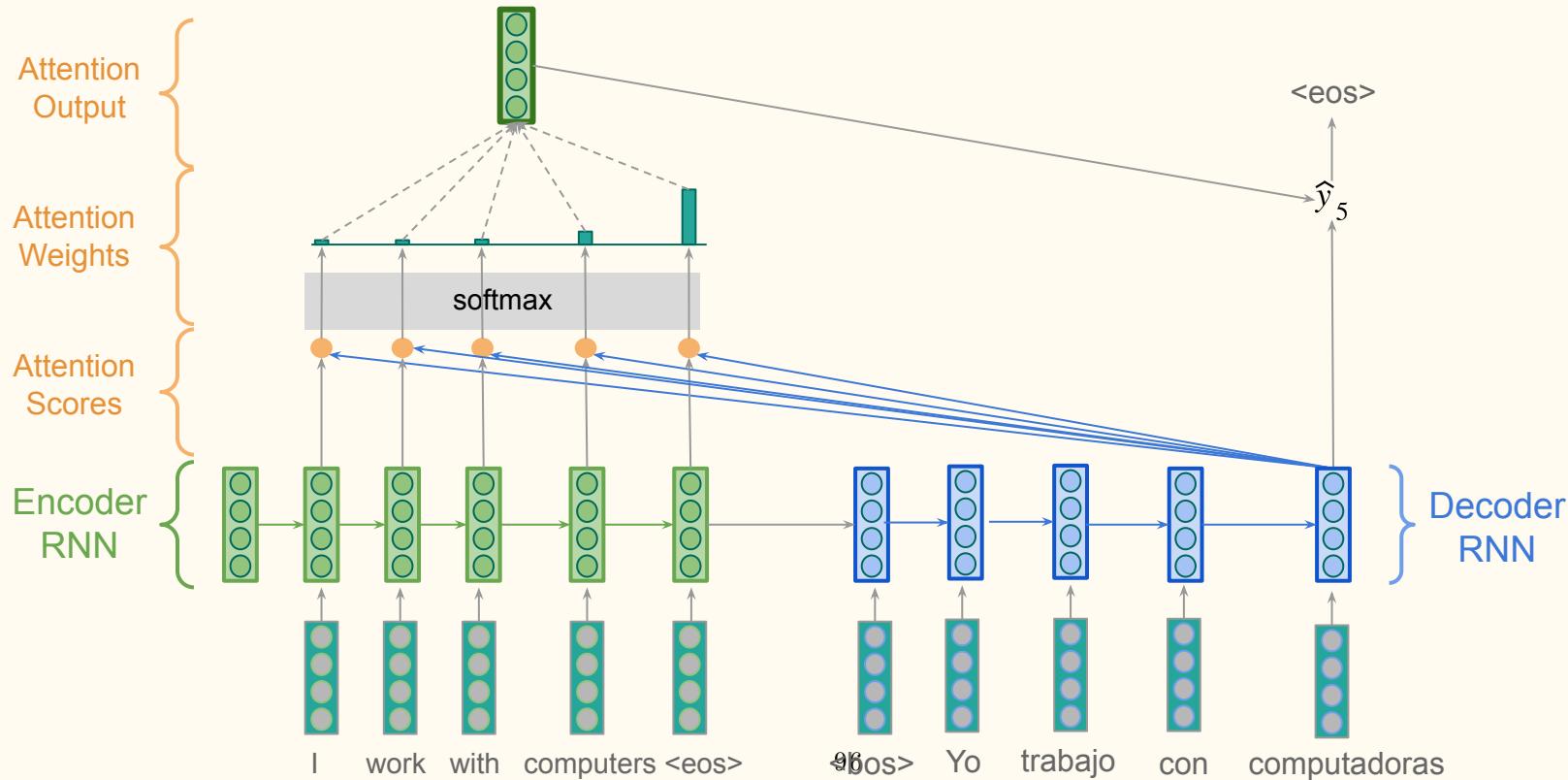
# Sequence-to-Sequence with Attention



# Sequence-to-Sequence with Attention



# Sequence-to-Sequence with Attention



# Computation Pipeline

- **Attention Input**

$$h_1, \dots, h_N$$

all encoder hidden states

$$s_t$$

decoder hidden state at timestep  $t$

- **Attention Scores**

$$\text{score}(s_t, h_k), k = 1..N$$

How do we compute  
score?

“How relevant is source token  $k$  for target step  $t$ ? ”

- **Attention Weights**

$$\alpha_k^{(t)} = \frac{\exp(\text{score}(s_t, h_k))}{\sum_{i=1}^N \exp(\text{score}(s_t, h_i))}, k = 1..N$$

- **Attention Output**

$$c^{(t)} = \alpha_1^{(t)} h_1 + \alpha_2^{(t)} h_2 + \dots + \alpha_N^{(t)} h_N = \sum_{k=1}^N \alpha_k^{(t)} h_k$$

“source context for decoder step  $t$  ”

# Attention Score Functions

Several ways to compute  $score \in \mathbb{R}^N$  from  $h_1, \dots, h_N \in \mathbb{R}^{d_1}$  and  $s \in \mathbb{R}^{d_2}$ :

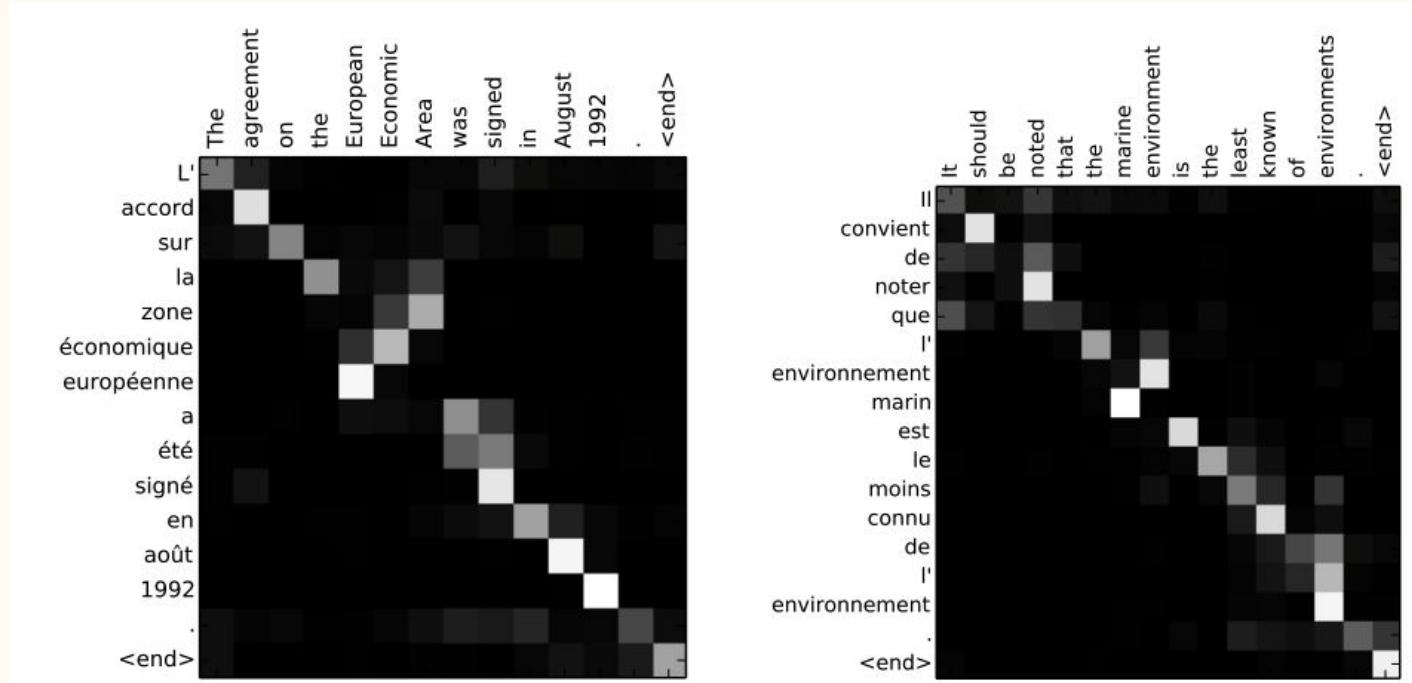
- **Dot-product Attention:**  $score_i = s^T h_i \in \mathbb{R}$
- **Multiplicative Attention:**  $score_i = s^T W h_i \in \mathbb{R}$
- **Additive Attention:**  $score_i = v^T \tanh(W_1 h_i + W_2 s) \in \mathbb{R}$

# Attention Score Functions

Several ways to compute  $score \in \mathbb{R}^N$  from  $h_1, \dots, h_N \in \mathbb{R}^{d_1}$  and  $s \in \mathbb{R}^{d_2}$ :

- **Dot-product Attention:**  $score_i = s^T h_i \in \mathbb{R}$ 
  - “Similarity”
- **Multiplicative Attention:**  $score_i = s^T W h_i \in \mathbb{R}$ 
  - Learn how much importance to give to different parts when calculating “similarity”
- **Additive Attention:**  $score_i = v^T \tanh(W_1 h_i + W_2 s) \in \mathbb{R}$ 
  - Simple single layer neural network to learn the scoring function

# Attention provides some interpretability

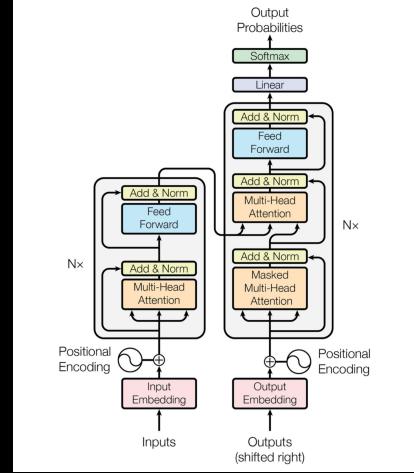


# Innovation

---

# The Transformer

(an overview)



# Motivation for Transformer

- Standard Neural Networks require fixed length input (so we need to pad sentences of different length)
  - Also input embeddings are not contextual
- RNNs allow us to get around this: We input one word at a time
  - Embeddings are contextual

# Motivation for Transformer

It was found that more layers and more pre-training helped language models

- RNNs need  $O(n)$  steps to process a sentence
  - As a consequence, we cannot pre-train on Very Large corpora
- The computational complexity per layer in RNNs is higher (when the input length is  $<$  the representation dimensionality)
  - As a consequence, we cannot have too many layers.

# Motivation for Transformer

What we need is an architecture that

- provides contextual embeddings
- captures semantic and syntactic information like RNNs do
- can process a sentence in parallel
- has relatively low computational complexity per layer

→ That's what the **Transformer** provides

# “Attention is All You Need”

	<b>Seq2Seq without attention</b>	<b>Seq2Seq with attention</b>	<b>Transformer</b>
processing within <i>encoder</i>	RNN	RNN	attention
processing within <i>decoder</i>	RNN	RNN	attention
<i>decoder</i> - <i>encoder</i> interaction	static fixed-sized vector	attention	attention

# Self-Attention

The self-attention for each word consists of a:

“Query” → Think of it as the question that tries to establish this word’s context.

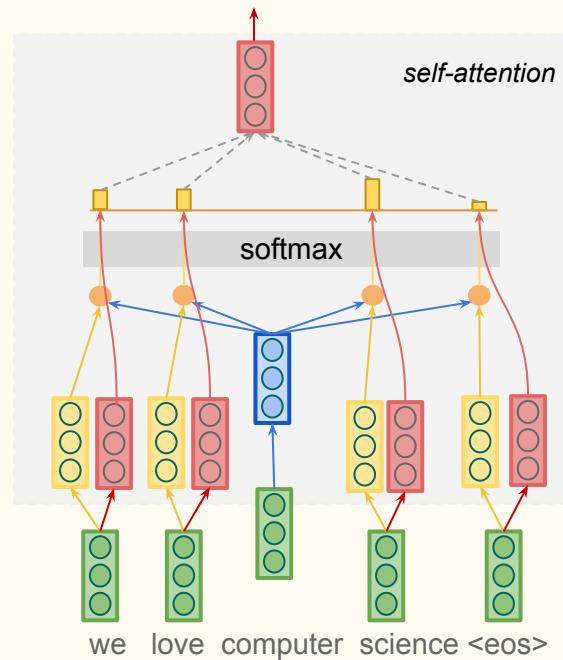
“Key” → (Query \* Key) gives the answer to how good *this* word is at providing context to the querying word (notice, this can be a different word)

“value” → The value of each word helping determine the answer to the query.

# Self-Attention: Query, Key, Value

Each vector receives a:

- **Query (q):** vector **from** which the attention is looking
- **Key (k):** vector **at** which the query looks to establish context
- **Value (v):** value of word being looked **at**, weighted based on context

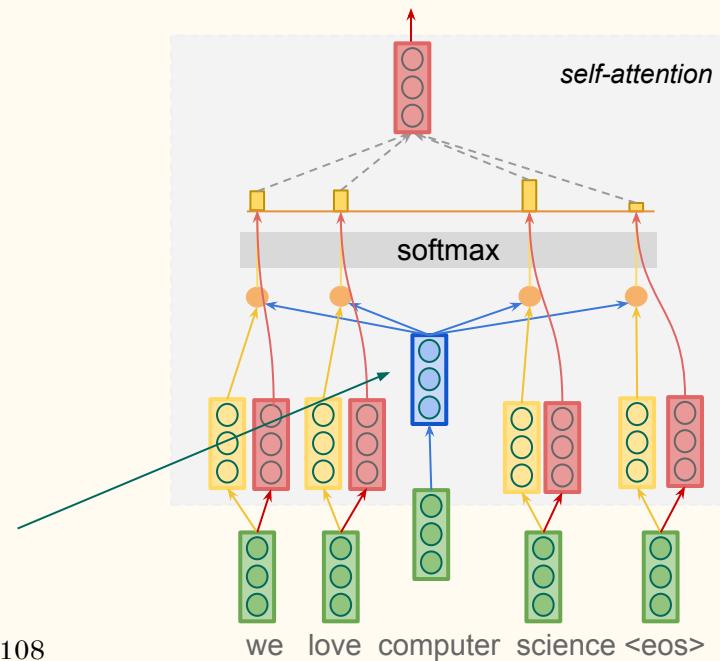


# Self-Attention: Query, Key, Value

Each vector receives a:

- **Query (q):** vector **from** which the attention is looking
- **Key (k):** vector **at** which the query looks to establish context
- **Value (v):** value of word being looked **at**, weighted based on context

The **Query** aims to establish the context, in this example, of the word “computer” (**from**)



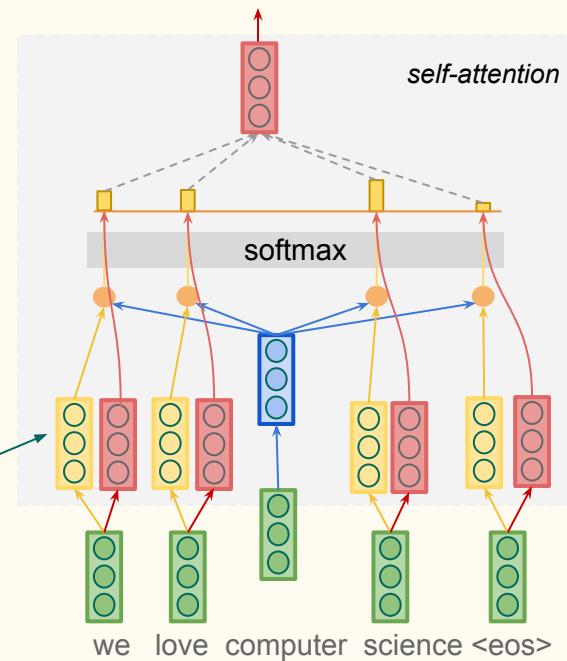
# Self-Attention: Query, Key, Value

Each vector receives a:

- **Query (q):** vector **from** which the attention is looking
- **Key (k):** vector **at** which the query looks to establish context
- **Value (v):** value of word being looked **at**, weighted based on context

The **Query** aims to establish the context, in this example, of the word “computer” (**from**)

Done by querying each **Key** of all words (**to**)

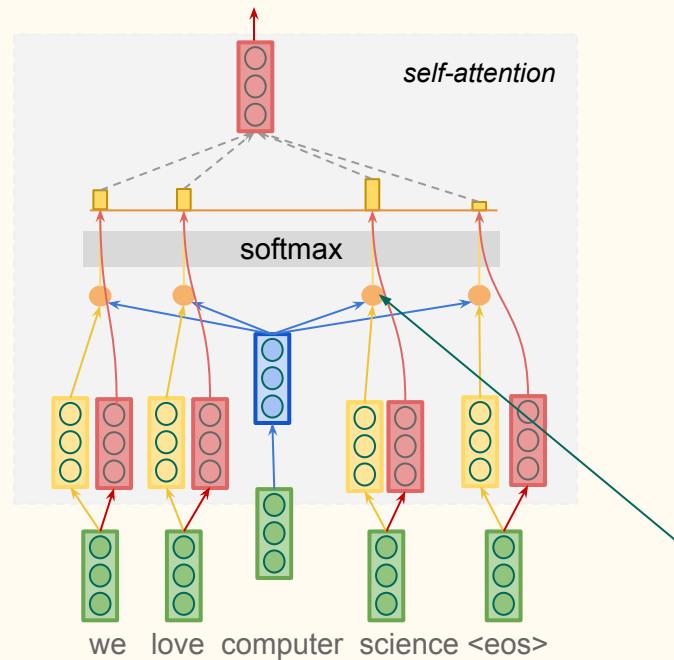


# Self-Attention: Query, Key, Value

Each vector receives a:

- **Query (q):** vector **from** which the attention is looking
- **Key (k):** vector **at** which the query looks to establish context
- **Value (v):** value of word being looked **at**, weighted based on context

In our example, this might establish “science” as the word that provides the most informative context.

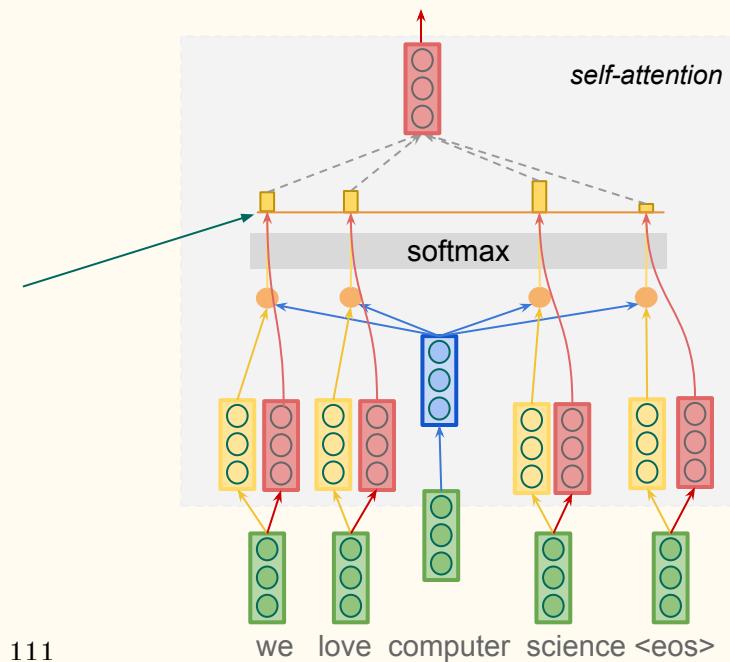


# Self-Attention: Query, Key, Value

Each vector receives a:

- **Query (q):** vector **from** which the attention is looking
- **Key (k):** vector **at** which the query looks to establish context
- **Value (v):** value of word being looked **at**, weighted based on context

We push the output of this through a softmax and multiply each with the **values** associated with each word (target word, i.e “at”)



# Self-Attention: Query, Key, Value

Each vector receives a:

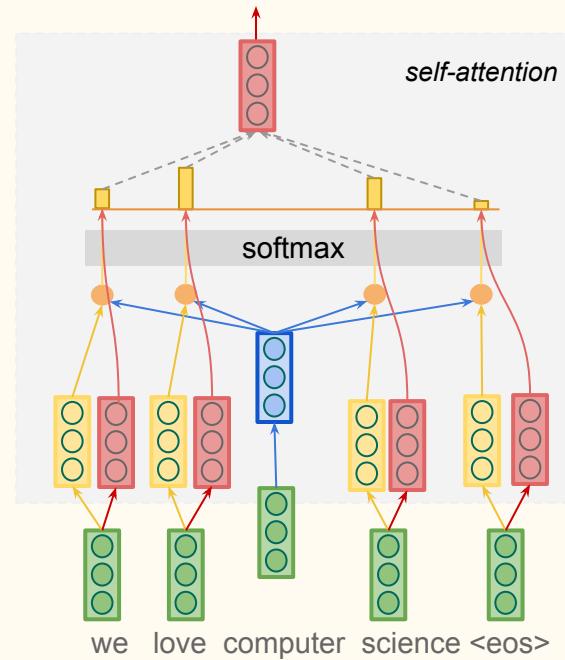
- **Query (q):** vector **from** which the attention is looking
- **Key (k):** vector **at** which the query looks to establish context
- **Value (v):** value of word being looked **at**, weighted based on context

$$\text{Attention}(q, k, v) = \text{softmax}\left(\frac{qk^T}{\sqrt{d_k}}\right)v$$

Attention Weights

Vector dimensionality of K, V

from      to



# Self-Attention: Query, Key, Value

Each vector receives a:

- **Query (q):** vector **from** which the attention is looking
- **Key (k):** vector **at** which the query looks to establish context
- **Value (v):** value of word being looked **at**, weighted based on context

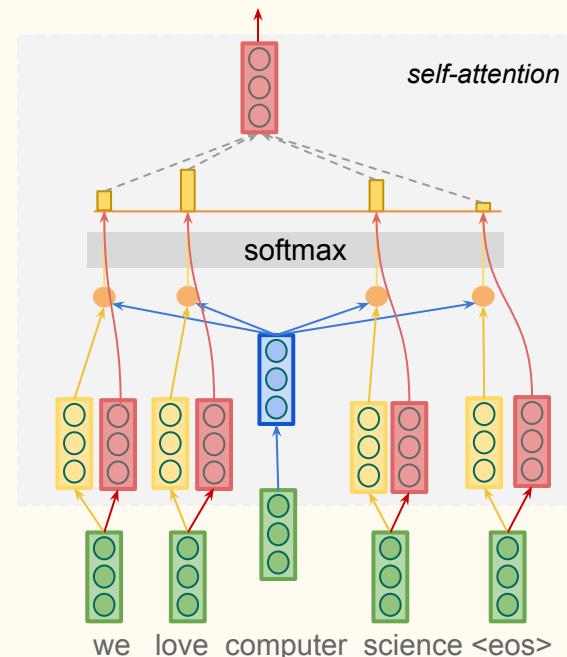
$$\text{Attention}(q, k, v) = \text{softmax}\left(\frac{qk^T}{\sqrt{d_k}}\right)v$$

Attention Weights

from      to

Used to make sure the value of  $qk^T$  does not become too large

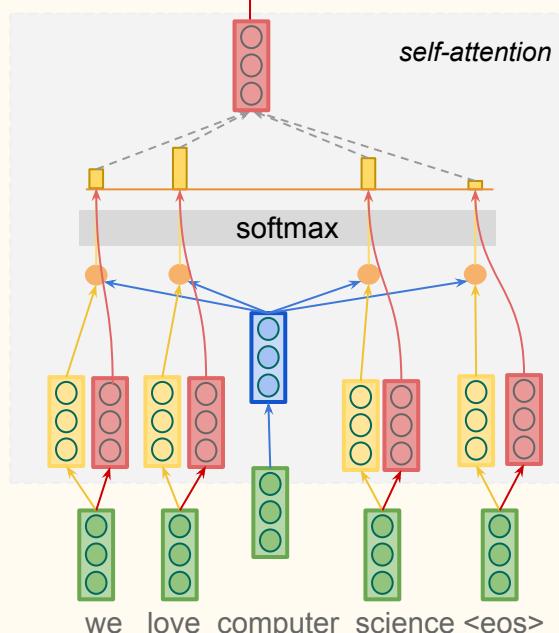
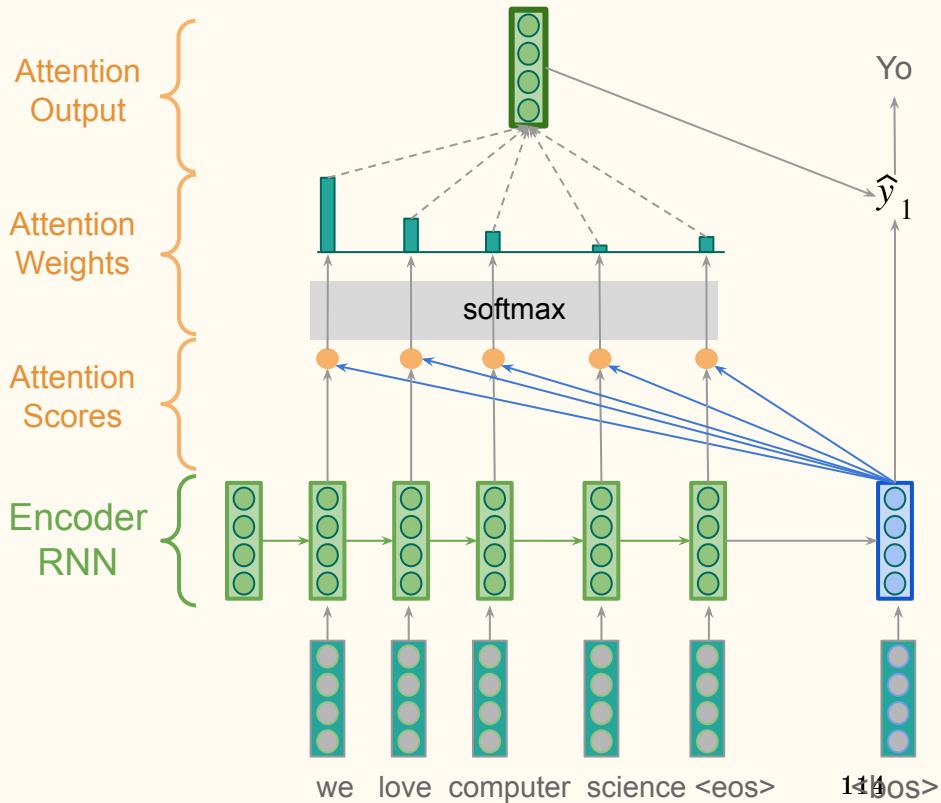
113



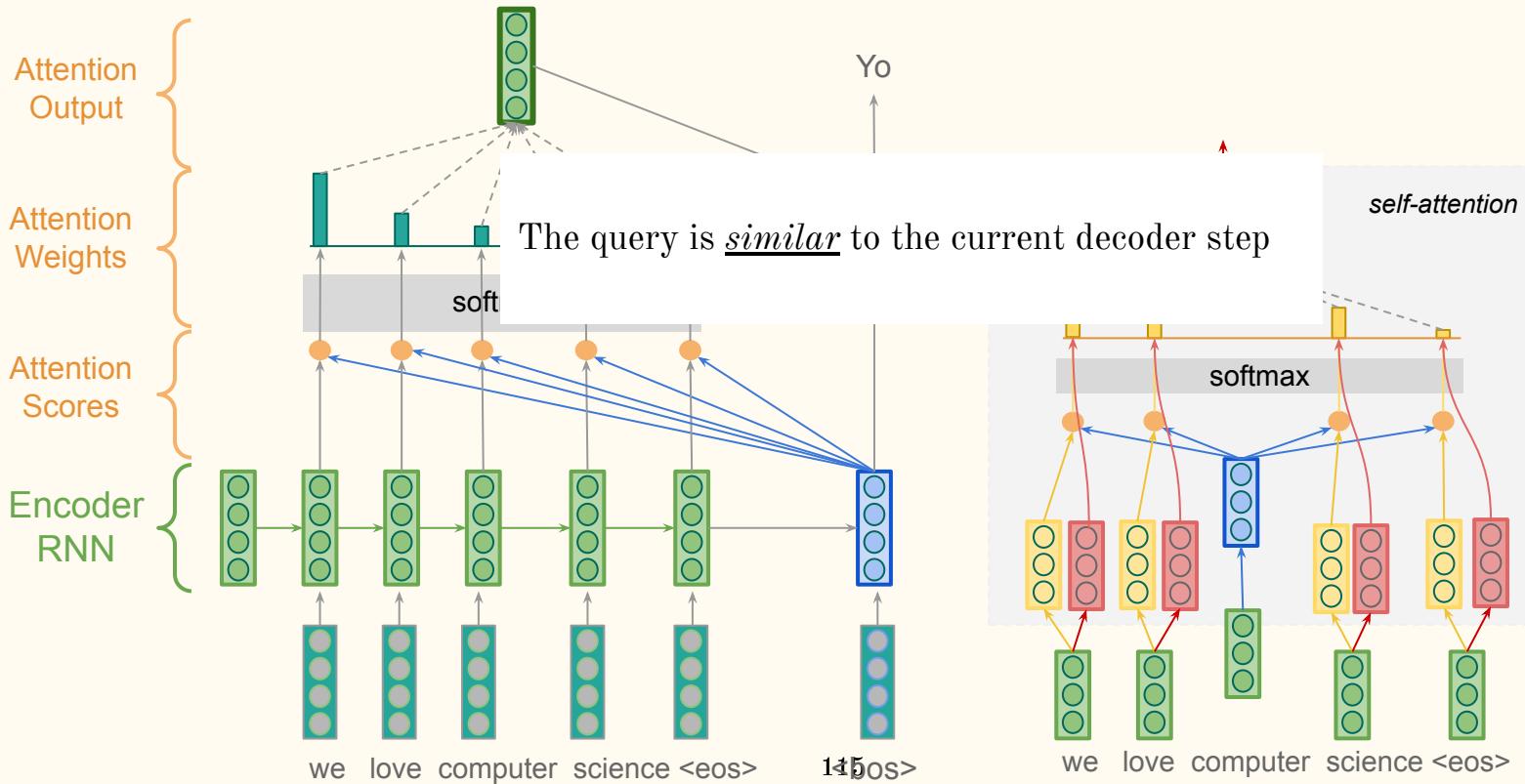
# Attention

vs

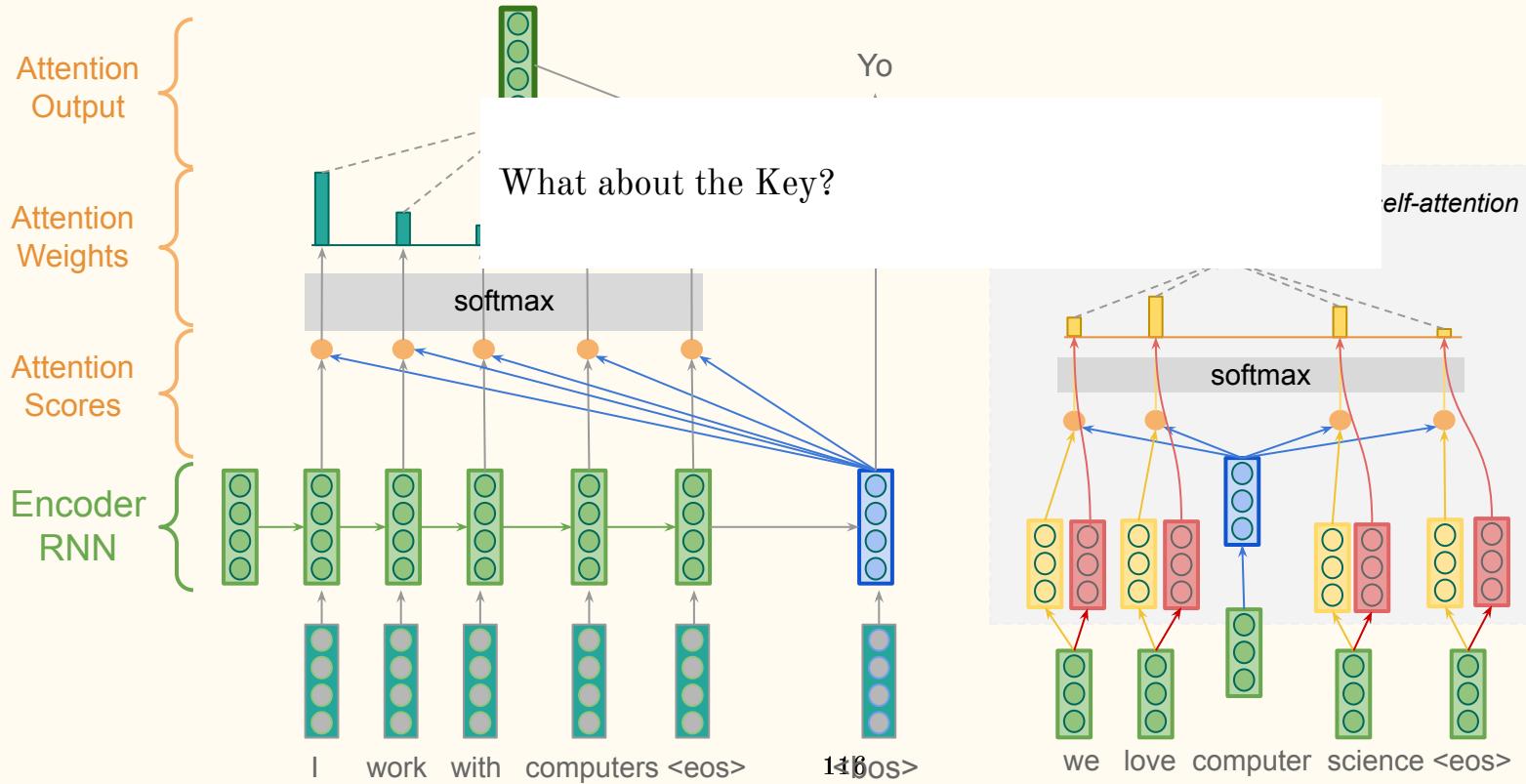
# Self-Attention



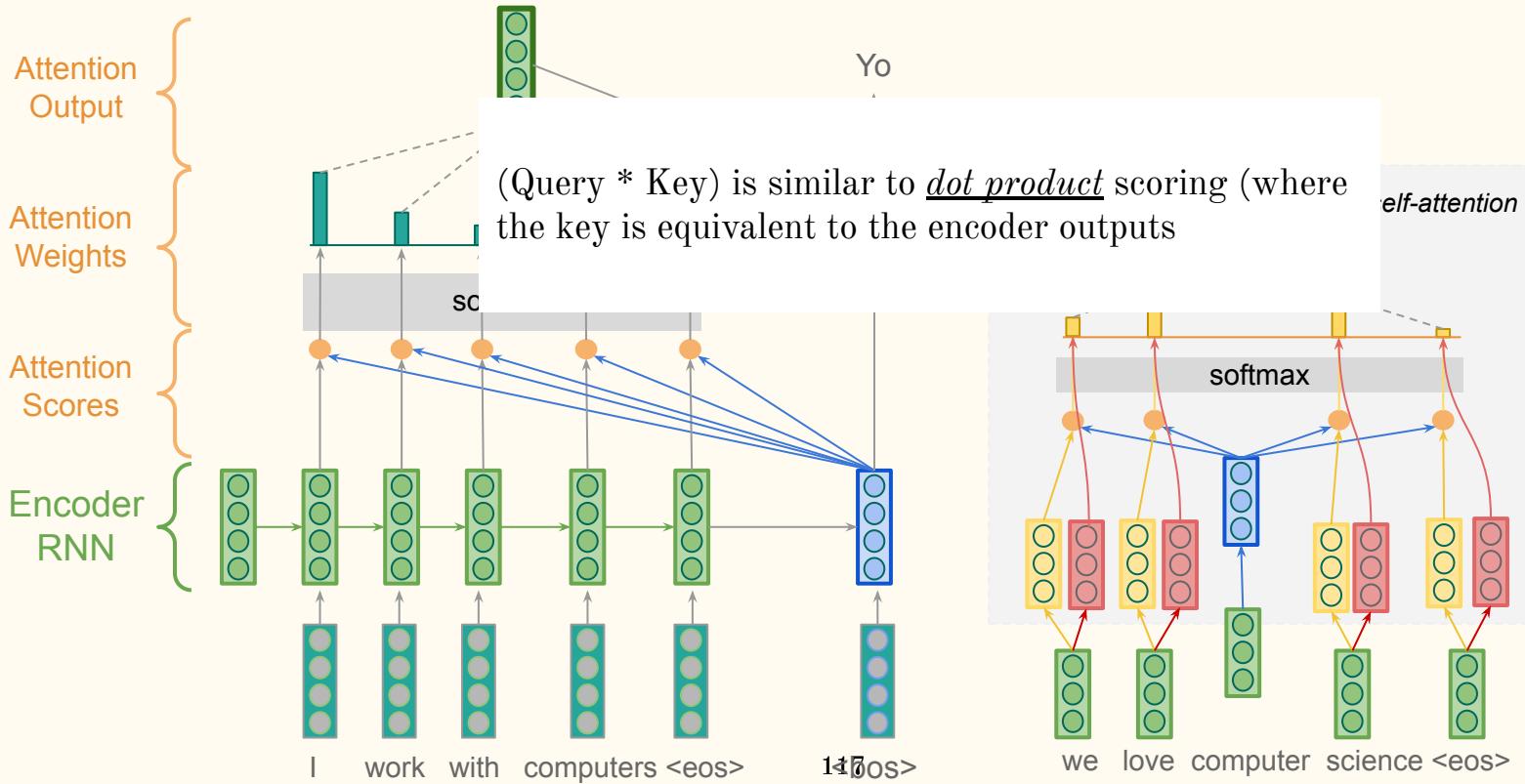
# Attention vs Self-Attention



# Attention vs Self-Attention

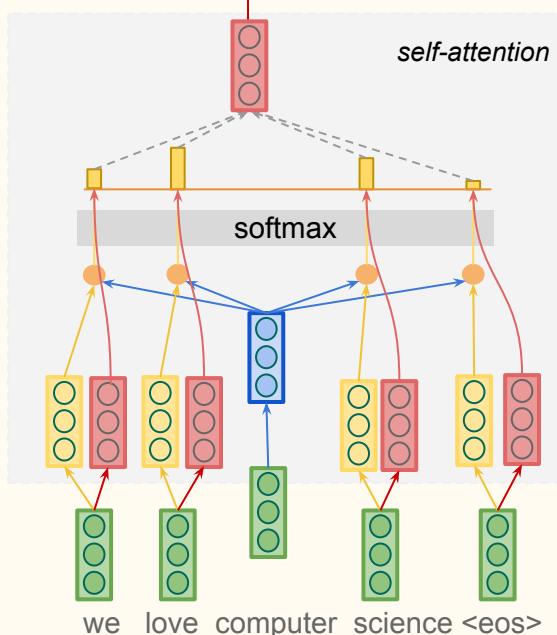
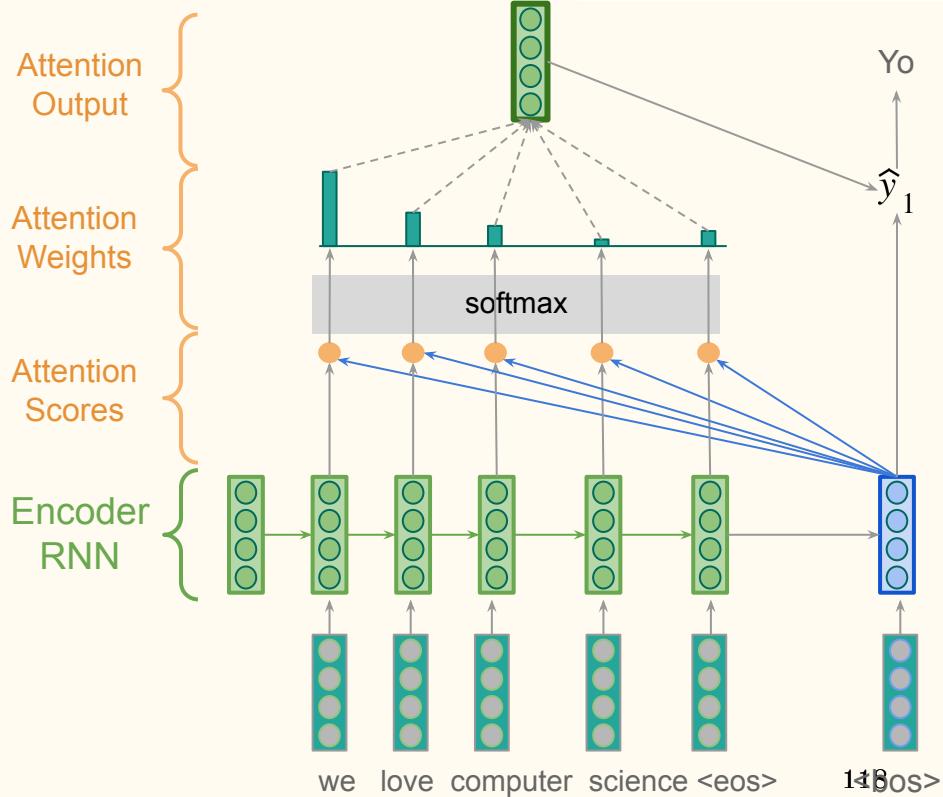


# Attention vs Self-Attention



And the Values?

# Attent



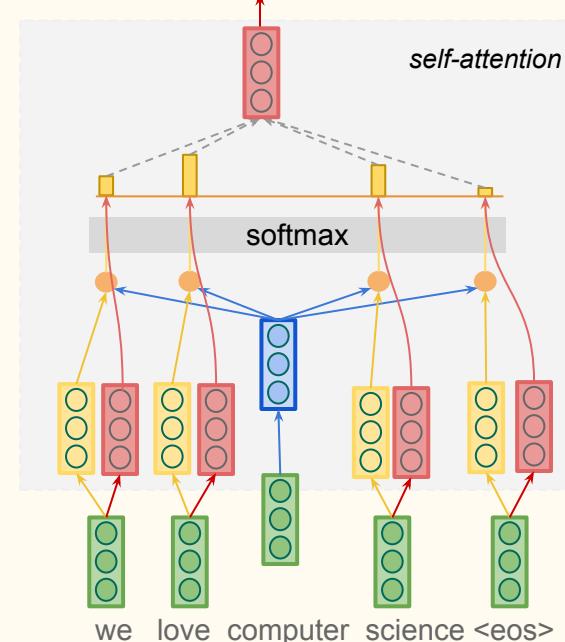
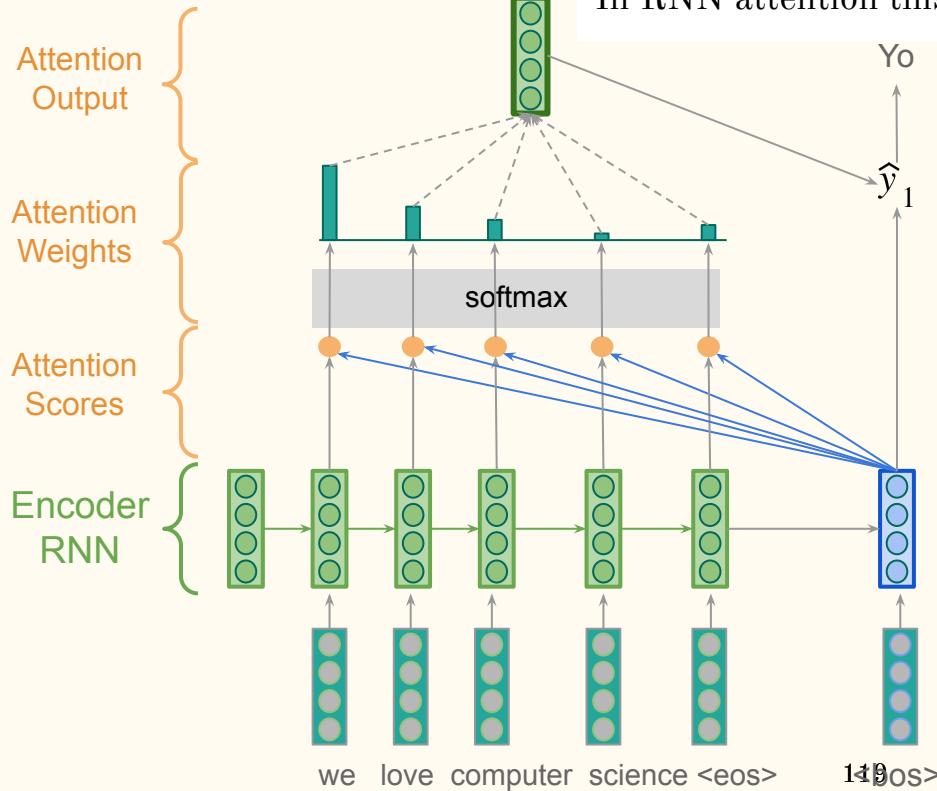
n

After pushing both through a softmax we multiply the result with the **value** in self-attention before summing it

n

# Attent

In RNN attention this is again the encoder states



# Attention

vs

# Self-Attention

$$\alpha_k^{(t)} = \frac{\exp(\text{score}(s_t, h_k))}{\sum_{i=1}^N \exp(\text{score}(s_t, h_i))}, \quad k = 1..N$$

$$A(q, K, V) = \sum_i \frac{\exp(q \cdot k^i)}{\sum_j \exp(q \cdot k^j)} \cdot v^i$$

Self attention ref:

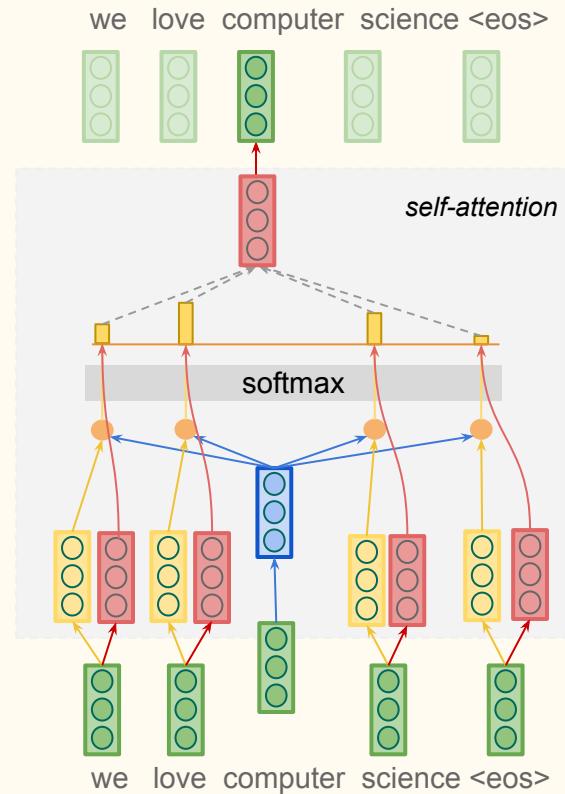
Andrew Ng <https://www.coursera.org/learn/nlp-sequence-models/lecture/lsvRK/self-attention>

# Self-Attention: Query, Key, Value

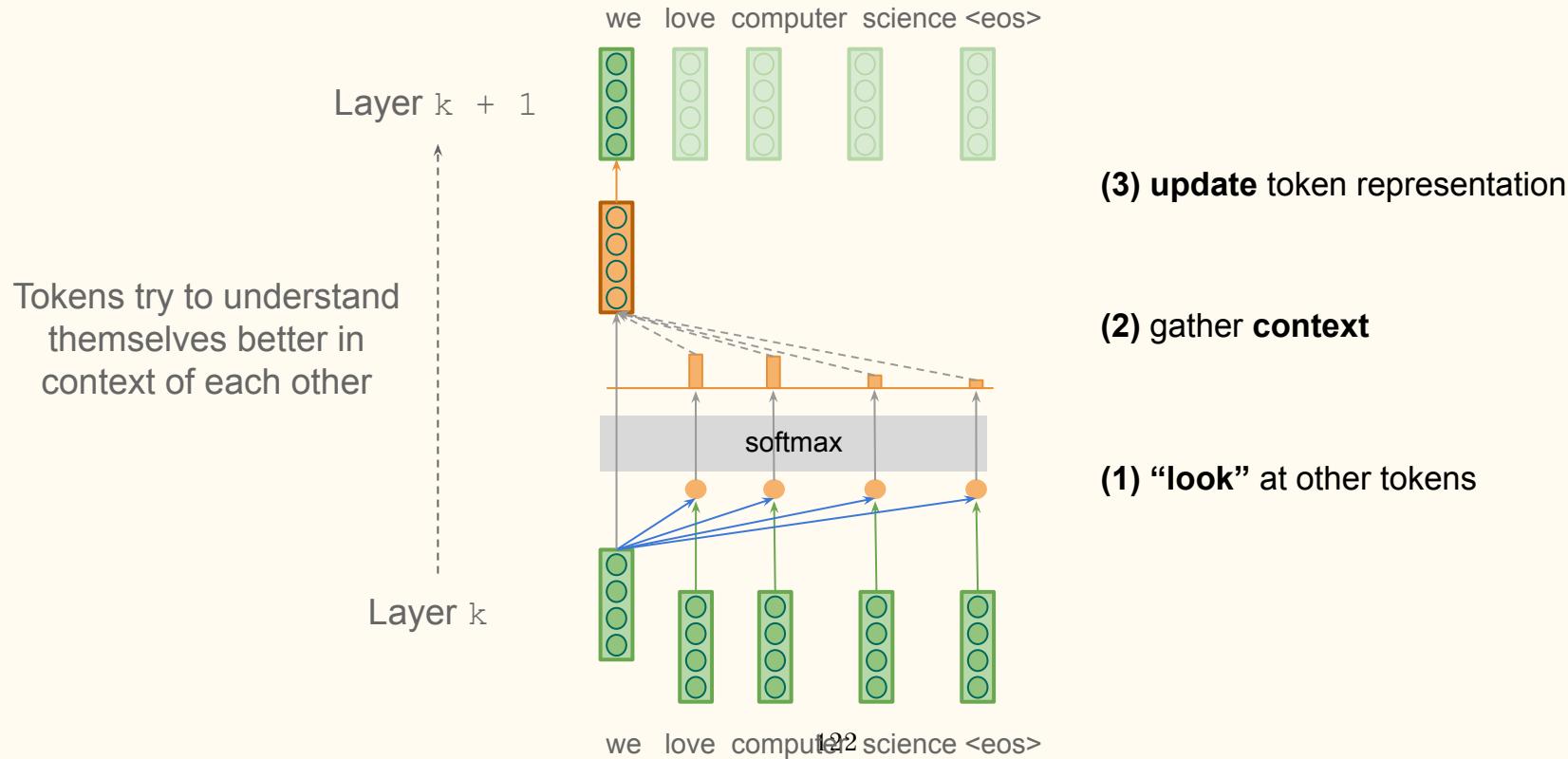
Each vector receives a:

- **Query (q):** vector **from** which the attention is looking
- **Key (k):** vector **at** which the query looks to establish context
- **Value (v):** value of word being looked **at**, weighted based on context

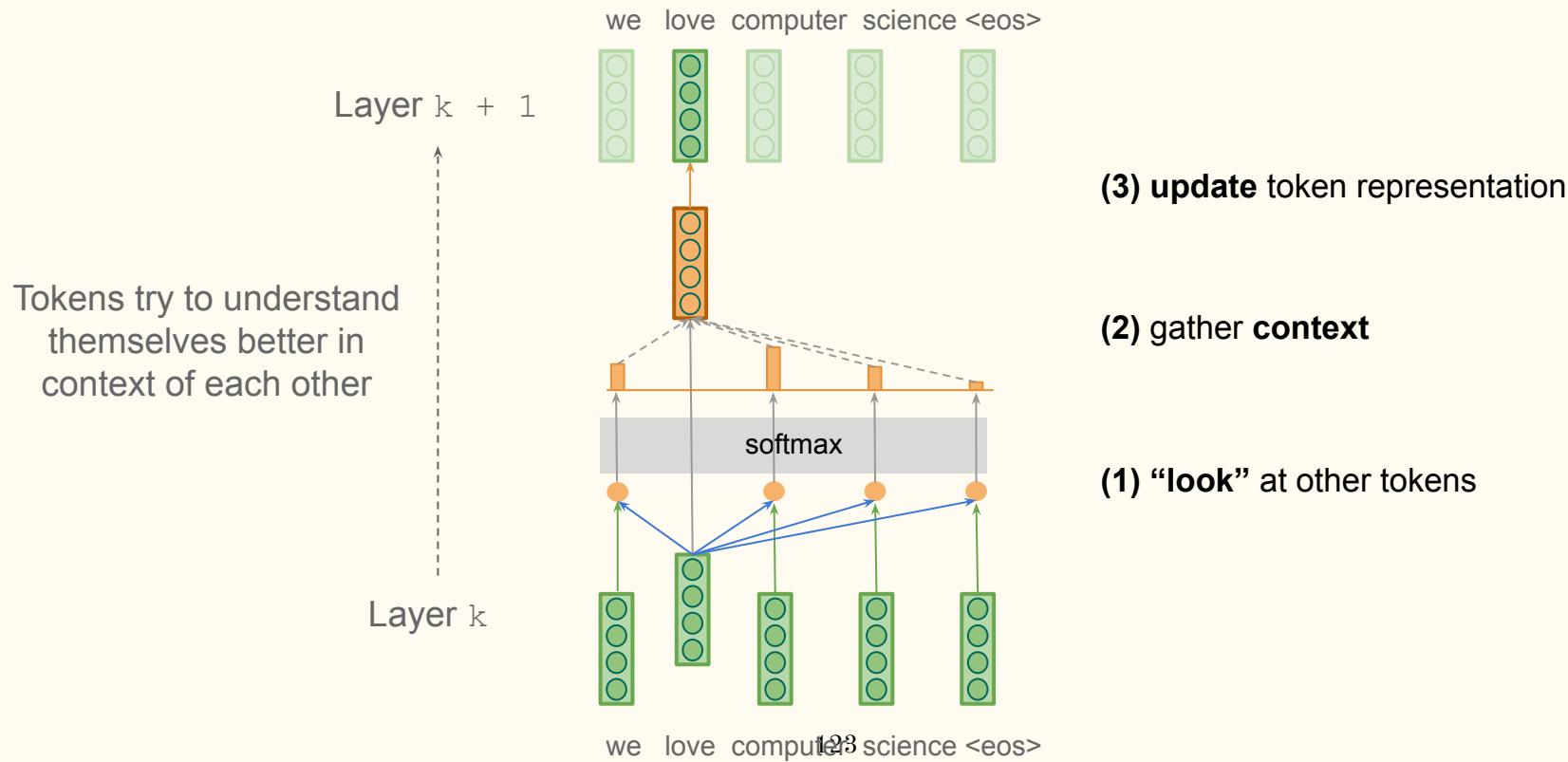
This is then fed to the next layer



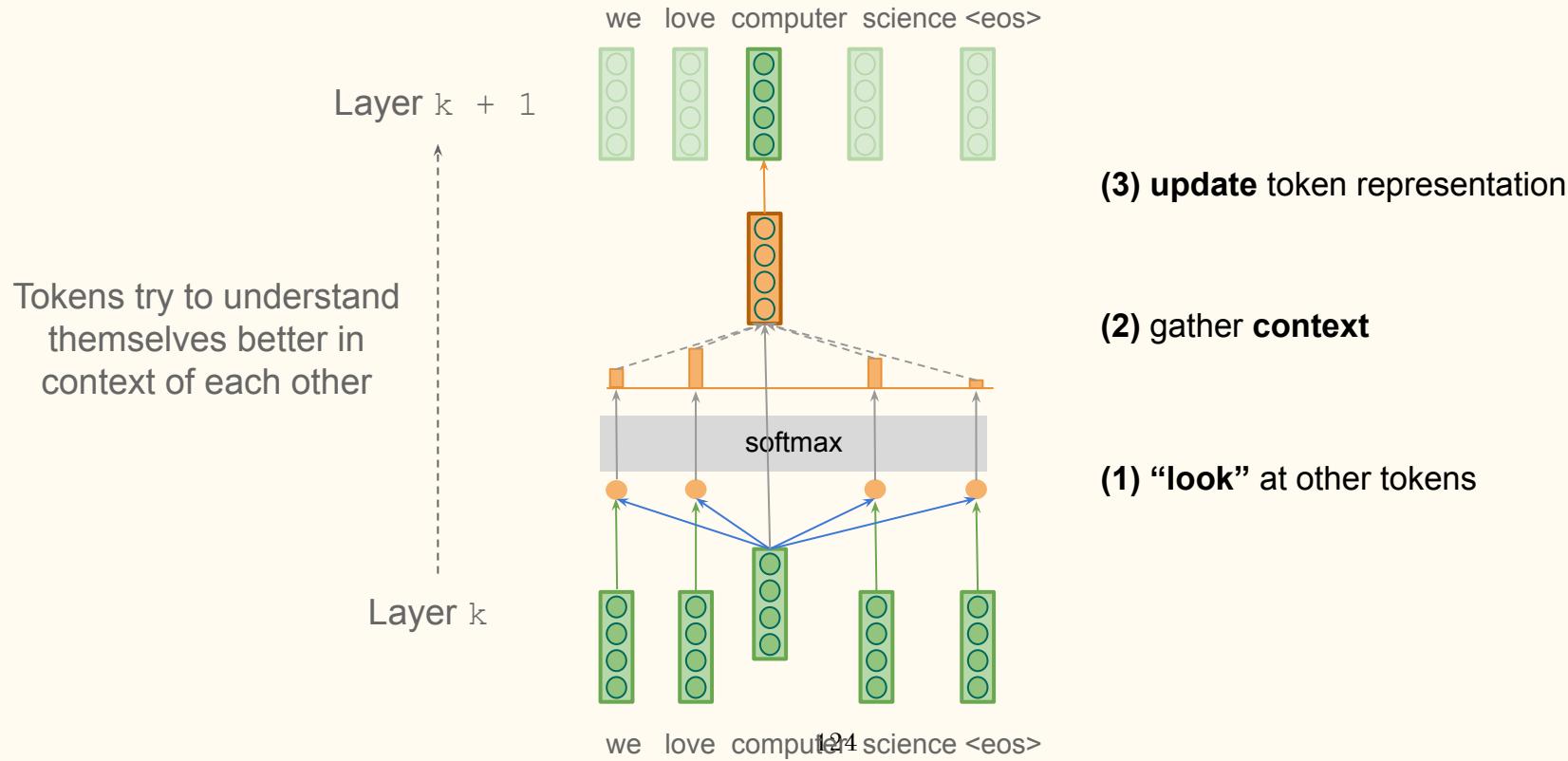
# Self-Attention



# Self-Attention



# Self-Attention

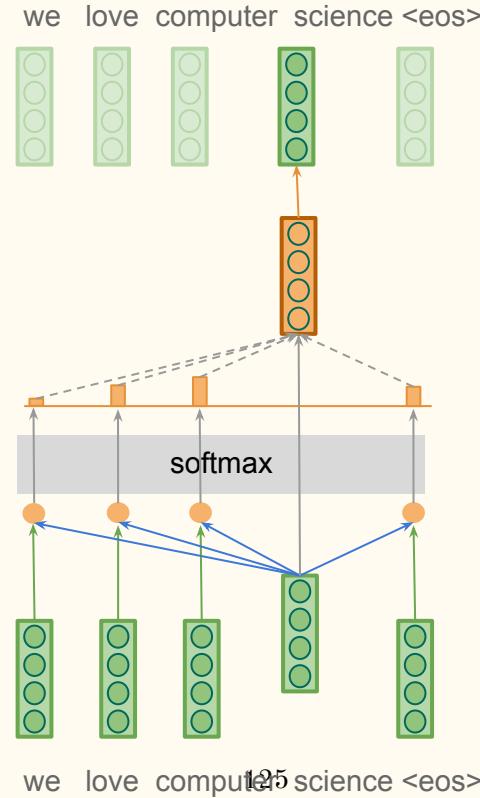


# Self-Attention

Tokens try to understand themselves better in context of each other

Layer  $k + 1$

Layer  $k$



**(3) update token representation**

**(2) gather context**

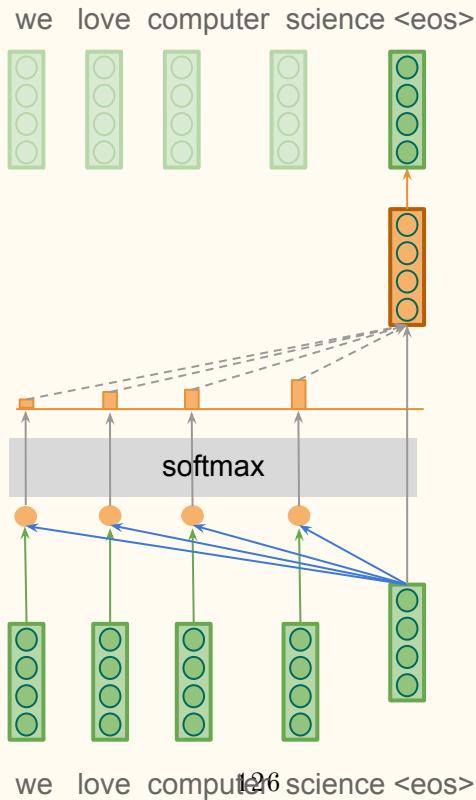
**(1) “look” at other tokens**

# Self-Attention

Tokens try to understand themselves better in context of each other

Layer  $k + 1$

Layer  $k$



In practice, this happens **in parallel**

**(3)** update token representation

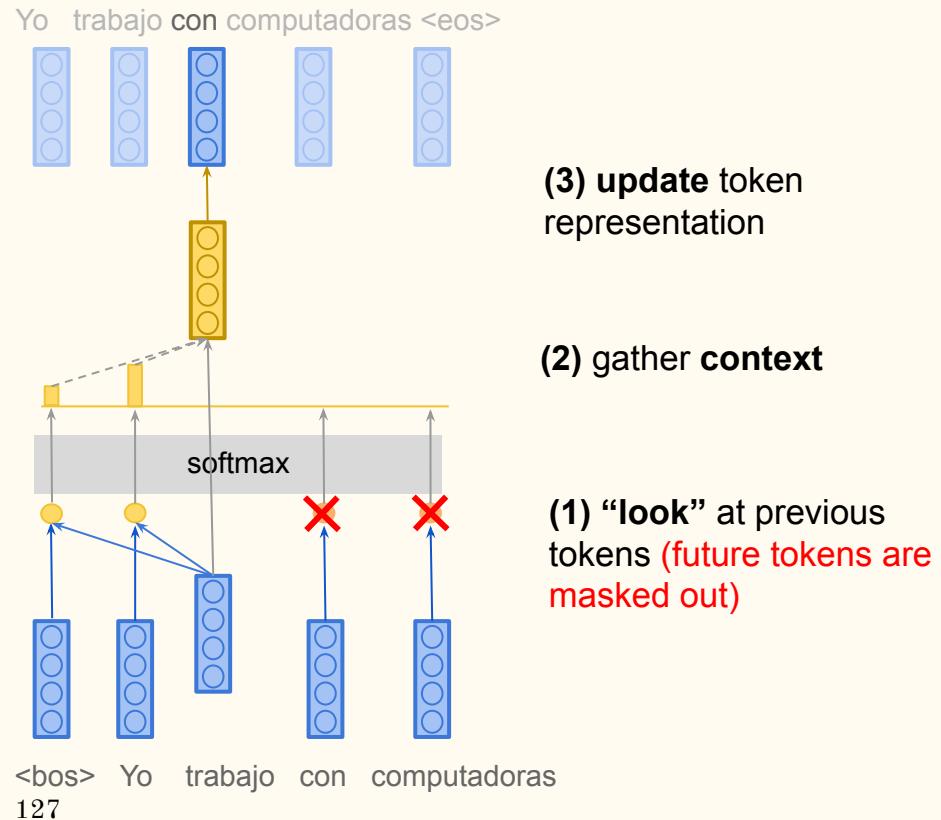
**(2)** gather context

**(1)** “look” at other tokens

# Masked Self-Attention

In the **Decoder**:

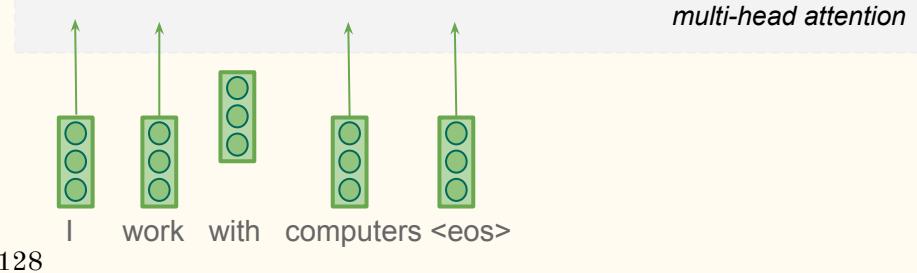
- During **inference**, we forbid looking at future tokens
  - It “doesn’t know them”
- During **training**, it can process all targets (without masks)
  - It can “see the future”



# Multi-Head Attention

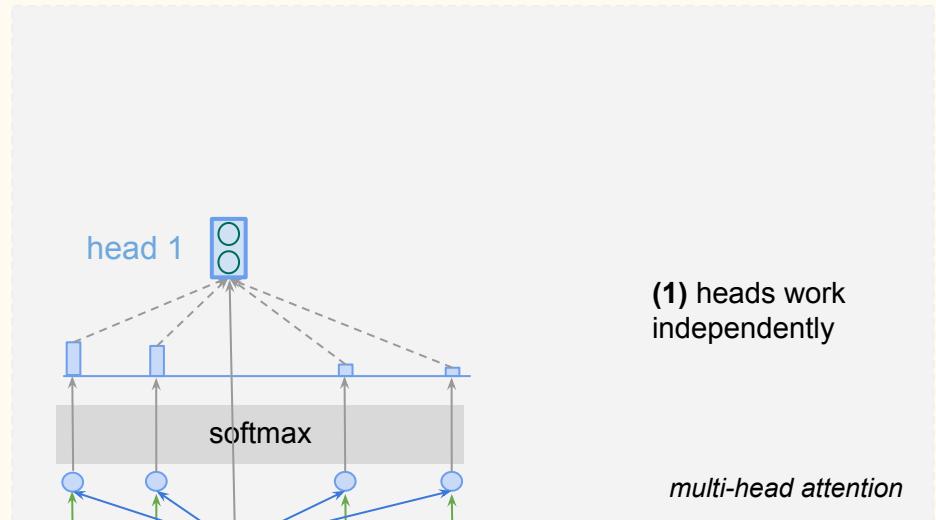
I work with computers <eos>  


(1) heads work independently



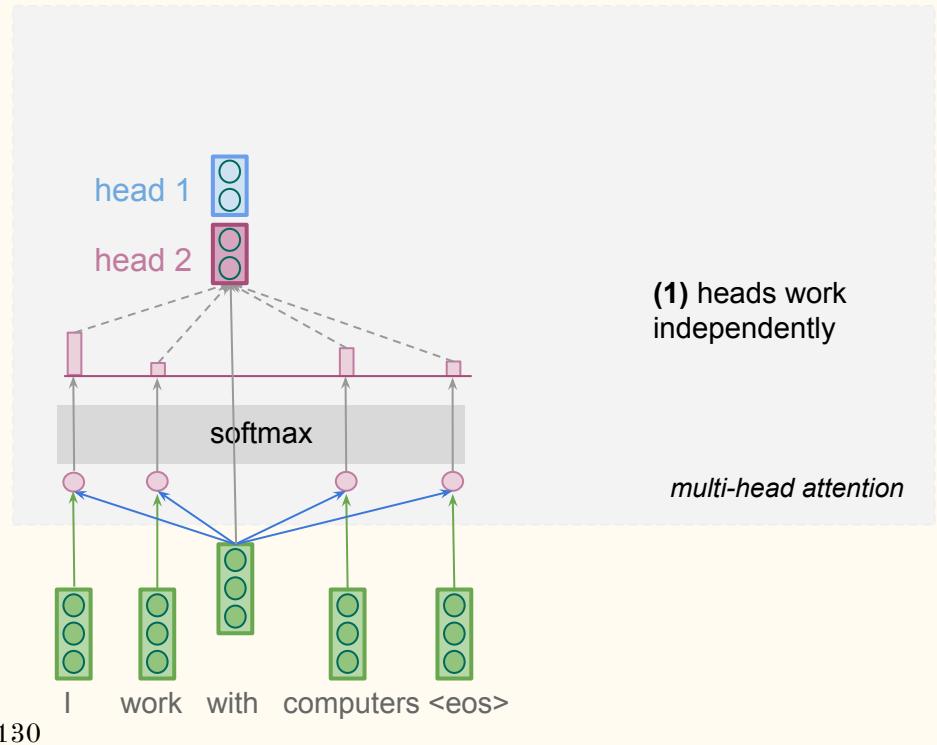
# Multi-Head Attention

I work with computers <eos>  

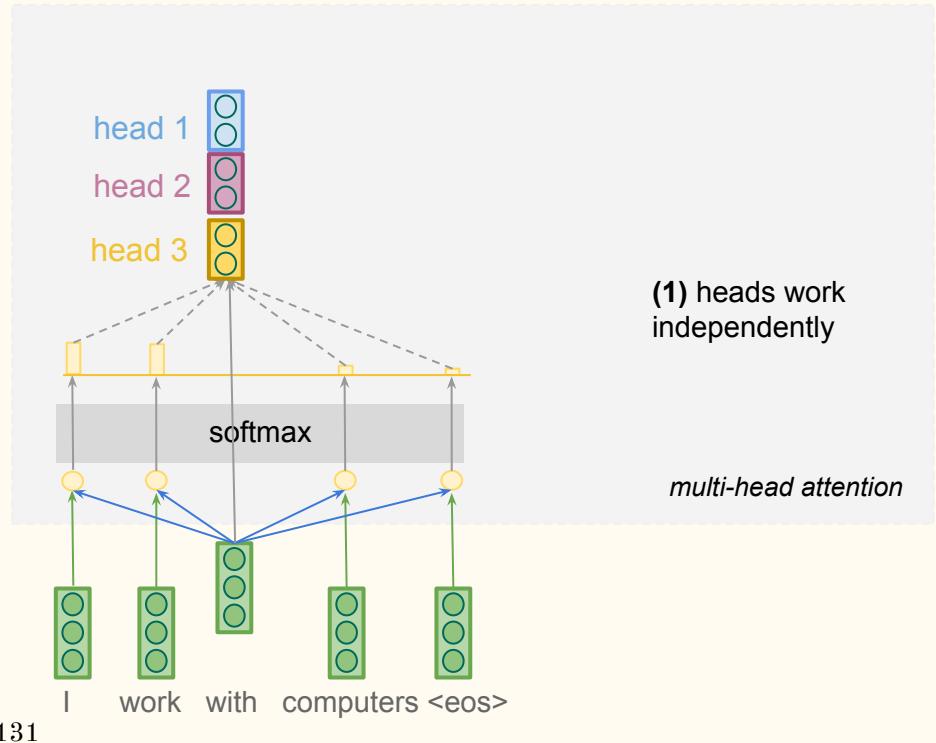
# Multi-Head Attention

I work with computers <eos>



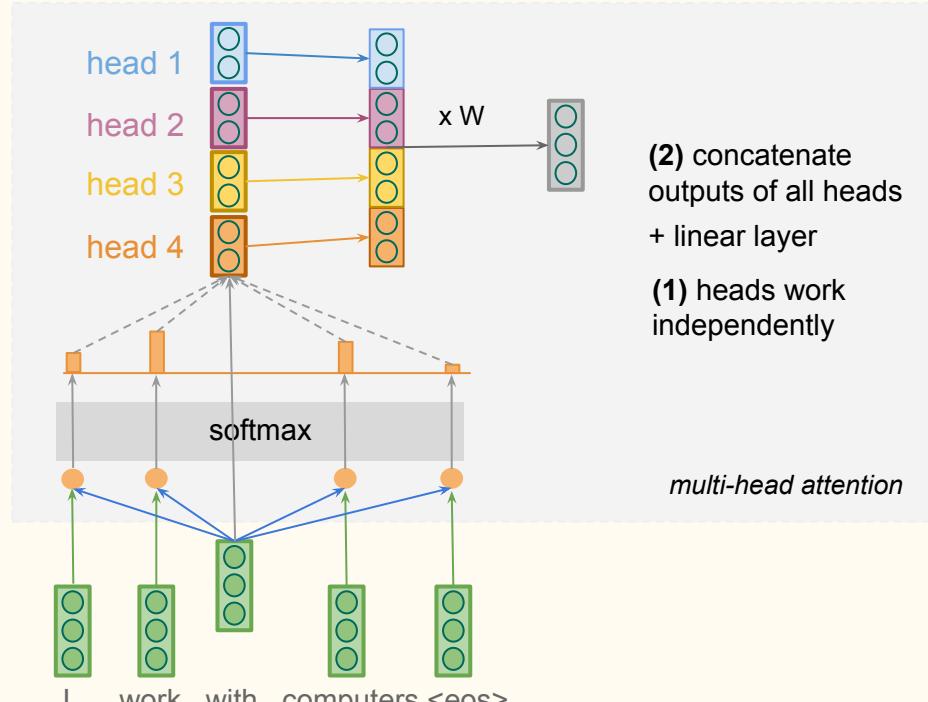
# Multi-Head Attention

I work with computers <eos>

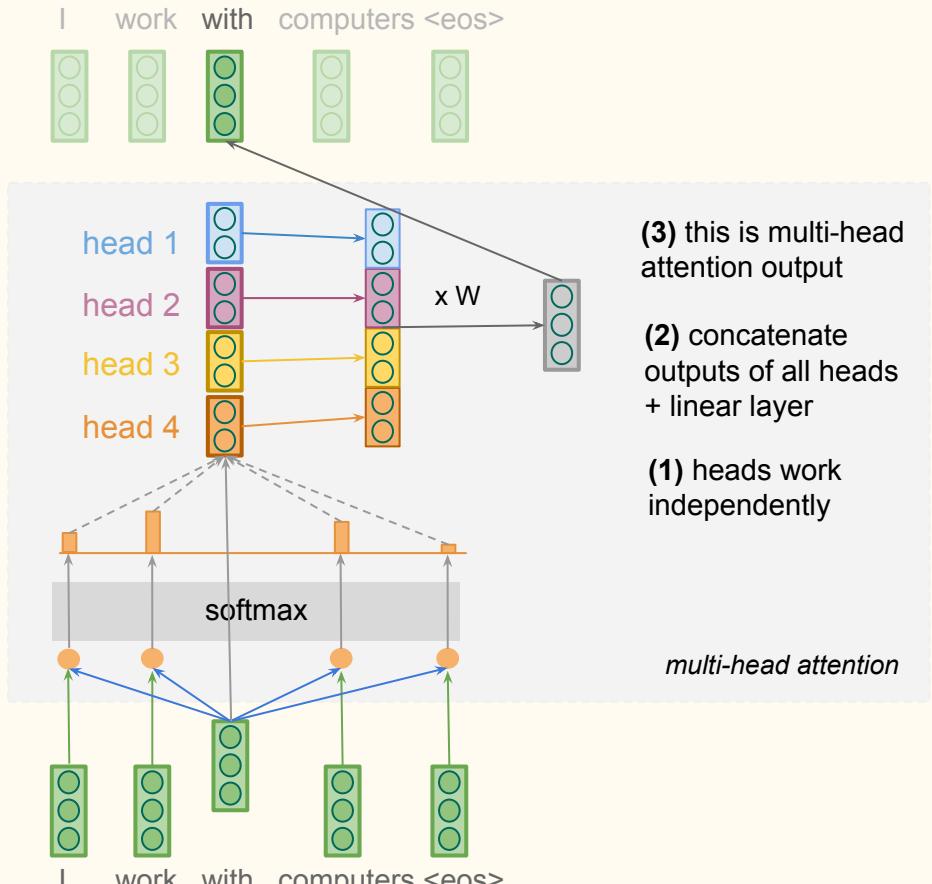


# Multi-Head Attention

I work with computers <eos>

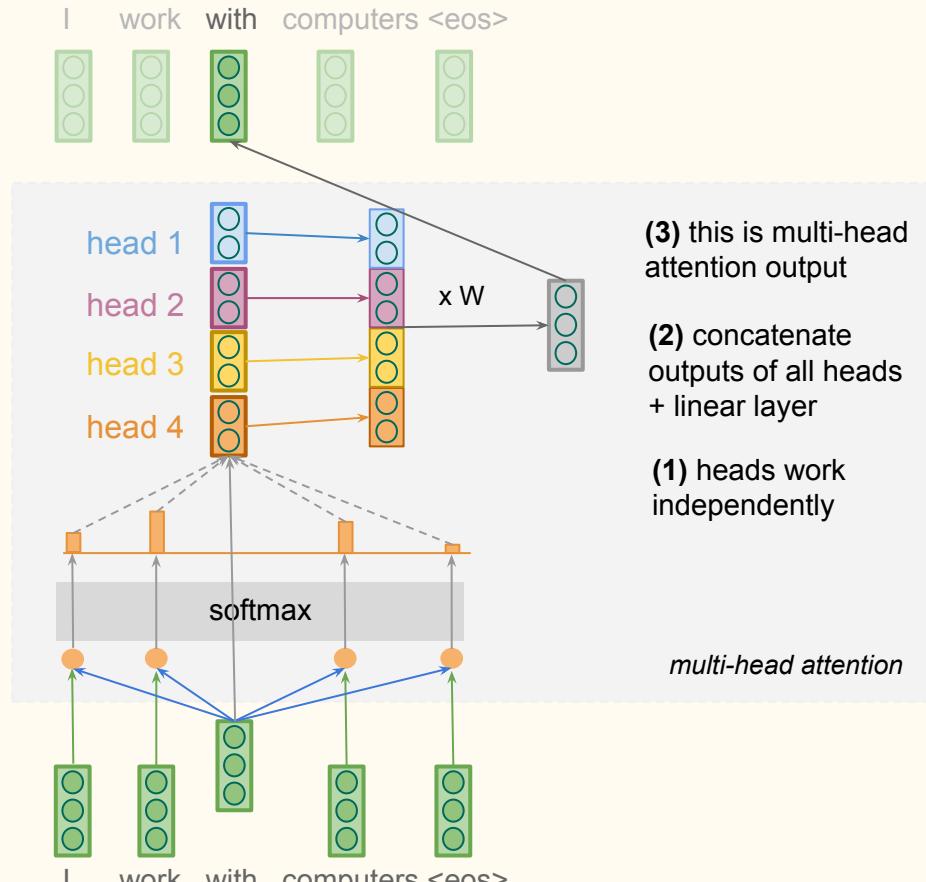


# Multi-Head Attention

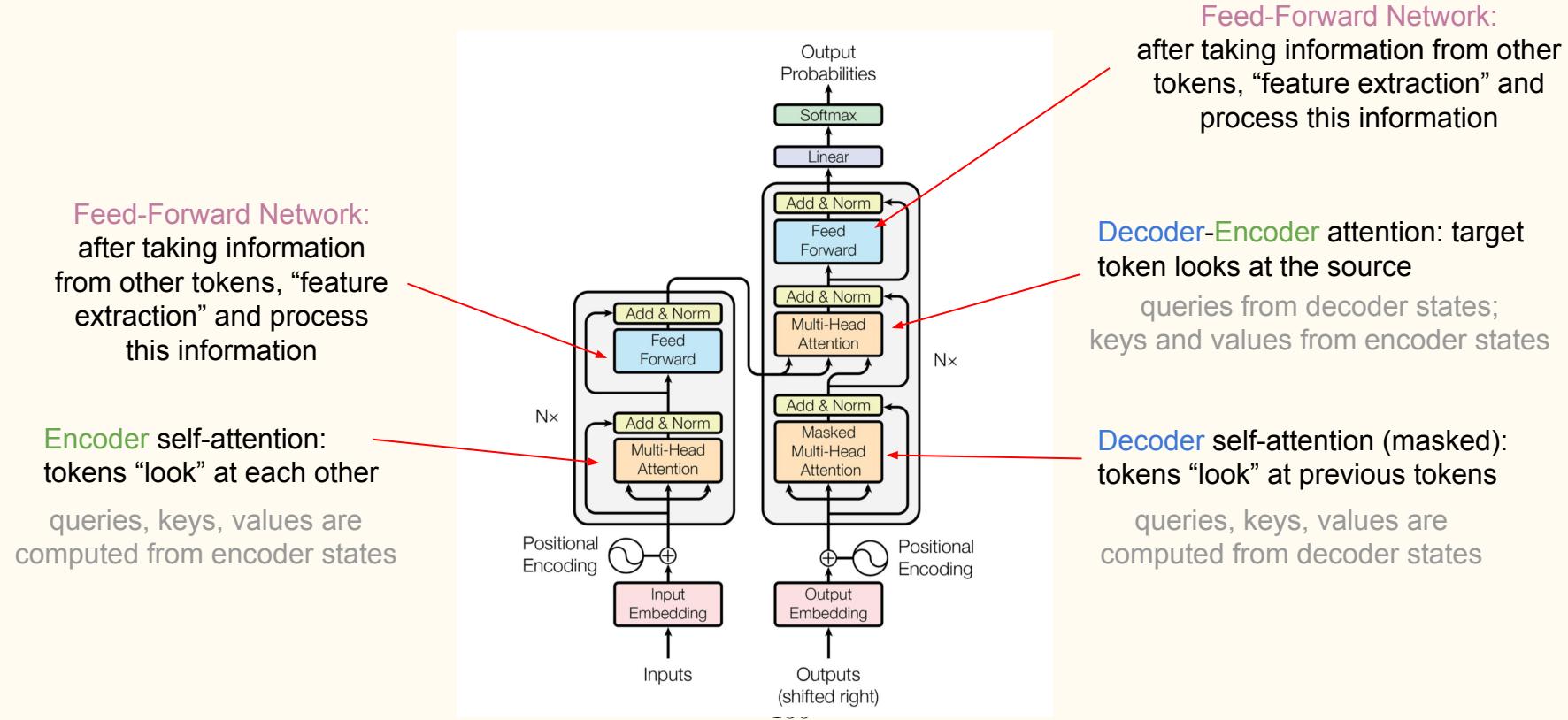


# Multi-Head Attention

MultiHead( $Q, K, V$ ) = Concat(head<sub>1</sub>, ..., head<sub>h</sub>) $W^O$   
where head<sub>i</sub> = Attention( $QW_i^Q, KW_i^K, VW_i^V$ )



# Transformer (Vaswani et al., 2017)



# Transformer (Vaswani et al., 2017)

- Positional Encoding
- Masked Self-Attention during training.
- Byte-Pair Encoding for subwords

For details on the other components, read the paper

# In this Section...

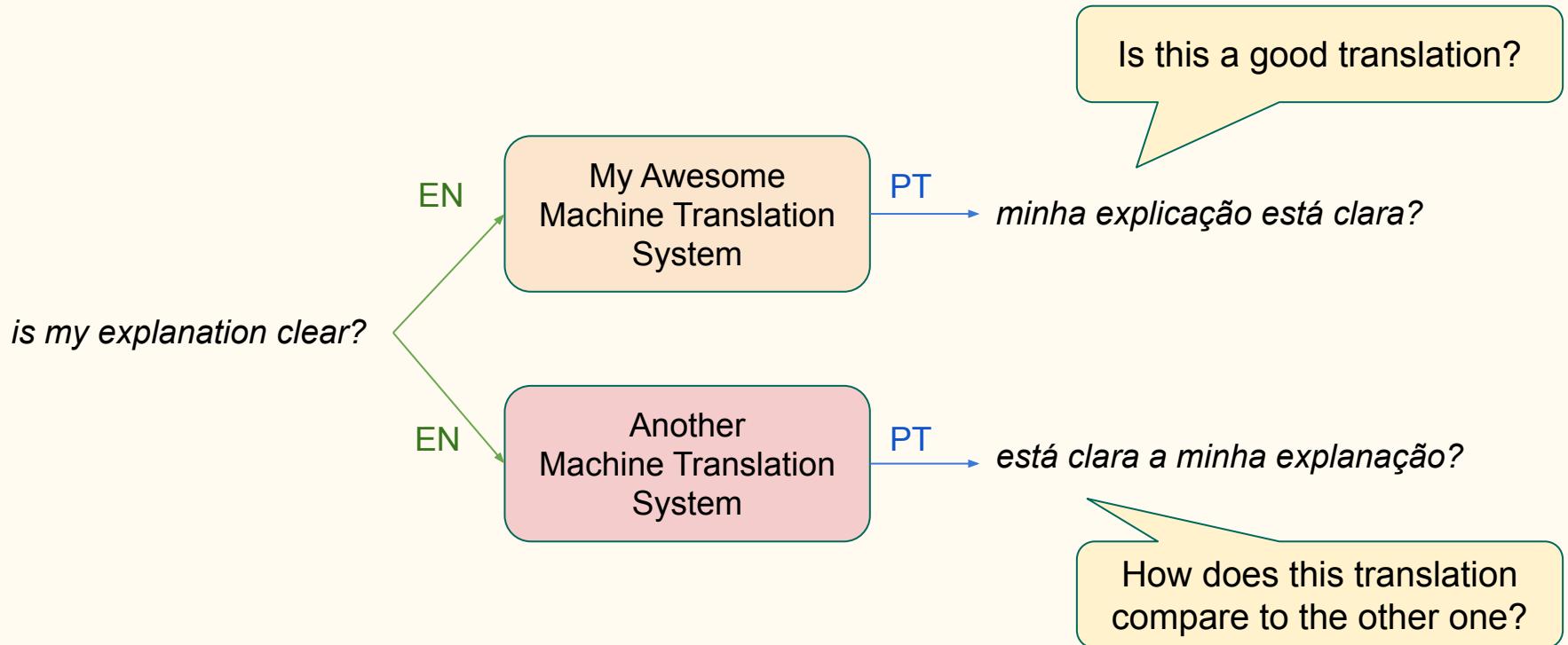
- Attention in Seq2Seq models with RNNs
  - Allows the model to focus on different parts of the source at different time steps during decoding
- The Transformer
  - “Attention is All You Need”
  - Self-Attention
    - For both Encoder and Decoder, “look around” to get better representations
  - Multi-Head Attention
    - Do not compute attention once, but several times and “combine” the representations (each can focus on different relevant aspects of the sentence)

# Evaluation

---

## Human Judgements and Automatic Metrics

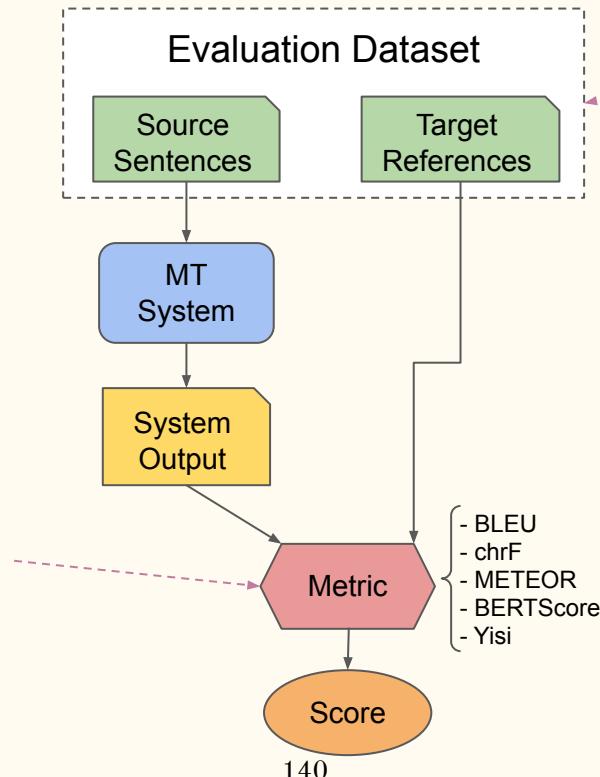
# How good is an MT output?



# Automatic Evaluation

For example, from WMT  
<http://www.statmt.org/wmt20/>

(Most) compute the “similarity”  
between the system output  
and the reference



# Automatic Metrics

Can be used in other Seq2Seq tasks!

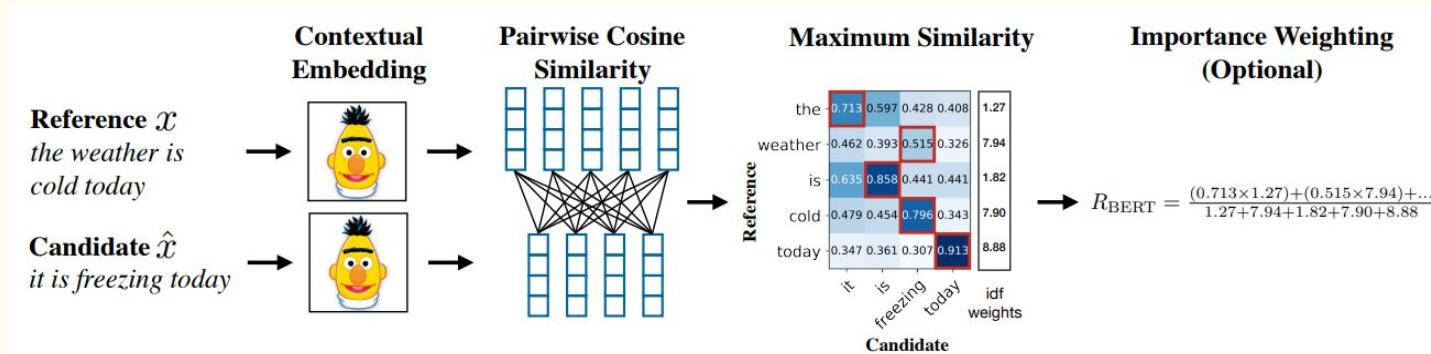
- BLEU (Papineni et al., 2002)

$$p_n = \frac{\sum_{S \in C} \sum_{ngram \in S} Count_{matched}(ngram)}{\sum_{S \in C} \sum_{ngram \in S} Count(ngram)}$$

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{1 - \frac{r}{c}} & \text{if } c \leq r \end{cases}$$

$$BLEU = BP \times \exp \left( \sum_{n=1}^N w_n \log p_n \right)$$

- BERTScore (Zhang et al., 2020)



# Human Judgements

- At **development time**, **automatic metrics** are useful because they are **fast to compute**, but they can be imprecise
- At **test time**, **manual evaluation** should also be performed
  - What to evaluate?
    - Fluency
    - Adequacy (a.k.a. meaning preservation)
  - How to evaluate?
    - Likert Scale (discrete score)
    - Direct Assessment (continuous score)
    - Relative Ranking

It's expensive!  
But **necessary** to truly  
evidence that the model is  
translating correctly

# Final Remarks

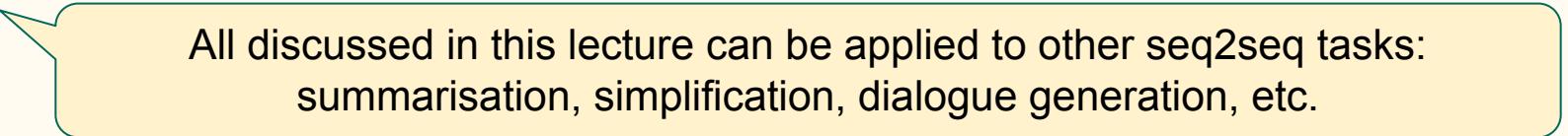
---

# Summary

- **Task:** Neural Machine Translation (NMT)
  - Given a source input sentence generate an output target translation
  - The machine translation model is a single neural network
- **Model:** Encoder-Decoder with RNNs
  - The encoder reads the source (token by token) and creates a representation of the whole sentence that is used by the decoder to generate the target translation
  - The encoder and the decoder are implemented using RNNs
- **Improvement:** Attention
  - Allows the decoder to focus on different parts of the source at different timesteps

# Summary

- **Improvement:** Transformer
  - For implementing the encoder, the decoder and their interaction it only relies on attention
  - Self-Attention allows to pay attention to all tokens in the same group at the same time
  - Computations can be parallelized, making it more efficient than RNN-based models
- **Evaluation:** Automatic Metrics and Manual Judgements
  - Serve to measure the quality of automatic translations
  - Automatic scores are useful at development time
  - Manual Judgements should be preferred at test time to validate the automatic scores



All discussed in this lecture can be applied to other seq2seq tasks:  
summarisation, simplification, dialogue generation, etc.

# Acknowledgements

- Ref from: Lecture 9, Lecture 10: Neural Machine Translation and Models with Attention
- Slides adapted from Fernando Alva Manchego
- Who, in turn, adapted slides from:
  - NLP Course | For You (by Lena Voita)
    - Sequence to Sequence (seq2seq) and Attention
  - Natural Language Processing with Deep Learning (Stanford CS 224N)
    - Lecture 7: Machine Translation, Sequence-to-Sequence and Attention

# Recommended Reading

- Blogs and Web Pages
  - [NLP Course | For You](#) (by Lena Voita)
  - [Attention and Augmented Recurrent Neural Networks](#)
  - [The Illustrated Transformer](#)
- Published Papers
  - [Neural Machine Translation by Jointly Learning to Align and Translate](#)
  - [Attention Is All You Need](#)

# Next Lecture

- **Transfer Learning**
  - Train a supervised model in one (or many) task(s) and re-use it in another
  - BERT → large Transformer-based language model

# References

- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. [Neural machine translation by jointly learning to align and translate](#). CoRR, abs/1409.0473.
- Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. [Bleu: A method for automatic evaluation of machine translation](#). In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, ACL '02, pages 311–318, Philadelphia, Pennsylvania. ACL.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. [Attention is all you need](#). In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems 30, pages 5998–6008. Curran Associates, Inc.
- Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. [Bertscore: Evaluating text generation with bert](#). In International Conference on Learning Representations.

# Transfer Learning for NLP

COM6513 Natural Language Processing

Nikos Aletras

n.aletras@sheffield.ac.uk

@nikaletras

Computer Science Department

Week 10  
Spring 2022



The  
University  
Of  
Sheffield.

In lectures 6 and 8...

- **Neural language modelling:** Probability of a word given some context

In lectures 6 and 8...

- **Neural language modelling:** Probability of a word given some context
  - Feedforward neural networks, e.g. skipgram

In lectures 6 and 8...

- **Neural language modelling:** Probability of a word given some context
  - Feedforward neural networks, e.g. skipgram
  - Recurrent neural networks, e.g. LSTM/GRU

## In lectures 6 and 8...

- **Neural language modelling:** Probability of a word given some context
  - Feedforward neural networks, e.g. skipgram
  - Recurrent neural networks, e.g. LSTM/GRU
- Neural LMs are trained on vast amounts of data

## In lectures 6 and 8...

- **Neural language modelling:** Probability of a word given some context
  - Feedforward neural networks, e.g. skipgram
  - Recurrent neural networks, e.g. LSTM/GRU
- Neural LMs are trained on vast amounts of data
- Labelled data is cheap, i.e. large publicly available corpora (aka **self supervision**)

## In lectures 6 and 8...

- **Neural language modelling:** Probability of a word given some context
  - Feedforward neural networks, e.g. skipgram
  - Recurrent neural networks, e.g. LSTM/GRU
- Neural LMs are trained on vast amounts of data
- Labelled data is cheap, i.e. large publicly available corpora (aka **self supervision**)
- Can we make use of this knowledge in downstream tasks where data might be scarce?

## In this lecture...

- **Transfer learning:** Re-use and adapt already pre-trained supervised machine learning models on a target task

## In this lecture...

- **Transfer learning:** Re-use and adapt already pre-trained supervised machine learning models on a target task
- How we can re-use neural LMs on target tasks (e.g. text classification, machine translation, question answering, etc.)

# Definition of Transfer Learning

A machine learning approach where models trained on a **source** task (or domain) are adapted to a related **target** task<sup>1</sup> (or domain)

---

<sup>1</sup>Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10), 1345-1359

## Definition of Transfer Learning (more formally)

Domain:  $\mathcal{D} = \{\mathcal{X}, P(X)\}$

Task:  $\mathcal{T}$  where  $y \in \mathcal{Y}$

Cond. Prob. Distrib.:  $P(Y|X)$

Given a source domain  $\mathcal{D}_S$  and a corresponding task  $\mathcal{T}_S$ , a target domain  $\mathcal{D}_T$  and task  $\mathcal{T}_T$ , learn a new model that computes the target conditional probability distribution  $P(Y_T|X_T)$  in  $\mathcal{D}_T$  given information from  $\mathcal{D}_S$  and  $\mathcal{T}_S$

# Transfer Learning Variants

- $\mathcal{X}_S \neq \mathcal{X}_T$ : Different feature spaces in source and target domains, e.g. documents written in different languages  
**(cross-lingual adaptation)**

# Transfer Learning Variants

- $\mathcal{X}_S \neq \mathcal{X}_T$ : Different feature spaces in source and target domains, e.g. documents written in different languages (**cross-lingual adaptation**)
- $P(X_S) \neq P(X_T)$ : Different marginal probability distributions in source and target domains, e.g. restaurant reviews vs electronic product reviews (**domain adaptation**)

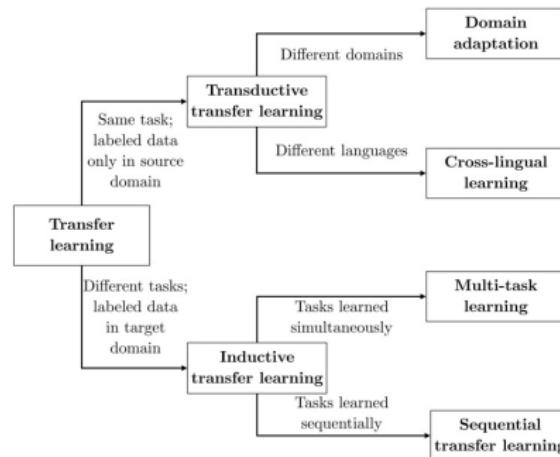
# Transfer Learning Variants

- $\mathcal{X}_S \neq \mathcal{X}_T$ : Different feature spaces in source and target domains, e.g. documents written in different languages (**cross-lingual adaptation**)
- $P(X_S) \neq P(X_T)$ : Different marginal probability distributions in source and target domains, e.g. restaurant reviews vs electronic product reviews (**domain adaptation**)
- $\mathcal{Y}_S \neq \mathcal{Y}_T$ : Different tasks (label sets), e.g. LM as source task and sentiment analysis as target task

# Transfer Learning Variants

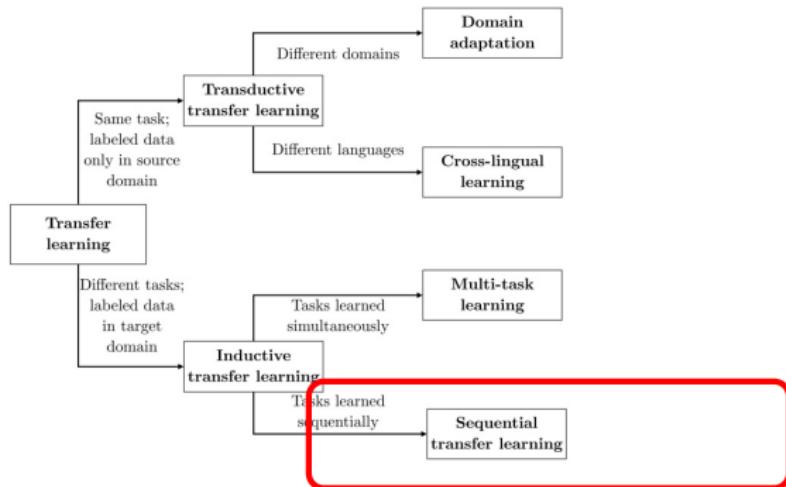
- $\mathcal{X}_S \neq \mathcal{X}_T$ : Different feature spaces in source and target domains, e.g. documents written in different languages (**cross-lingual adaptation**)
- $P(X_S) \neq P(X_T)$ : Different marginal probability distributions in source and target domains, e.g. restaurant reviews vs electronic product reviews (**domain adaptation**)
- $\mathcal{Y}_S \neq \mathcal{Y}_T$ : Different tasks (label sets), e.g. LM as source task and sentiment analysis as target task
- $P(Y_S|X_S) \neq P(Y_T|X_T)$ : Different conditional probability distributions between source and target tasks, e.g. source and target documents are unbalanced regarding to their classes

# Transfer Learning Taxonomy<sup>2</sup>



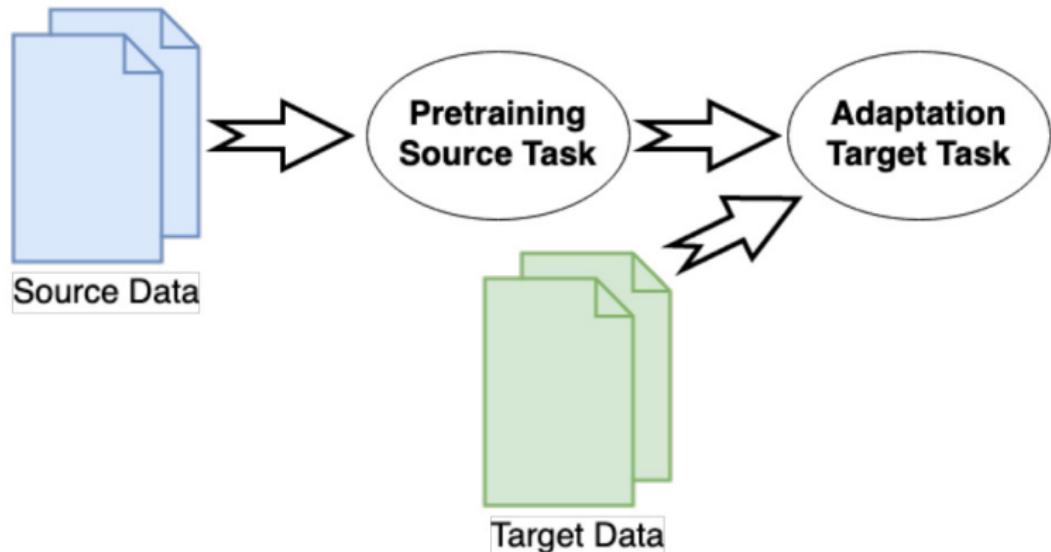
<sup>2</sup>Ruder, S. (2019). Neural transfer learning for natural language processing (Doctoral dissertation, NUI Galway)

# Transfer Learning Taxonomy<sup>2</sup>



<sup>2</sup>Ruder, S. (2019). Neural transfer learning for natural language processing (Doctoral dissertation, NUI Galway)

# Sequential Transfer Learning



# Pretraining: Language Modelling

- Source task: Language modelling

# Pretraining: Language Modelling

- Source task: Language modelling
- Cheap: no human annotations required

## Pretraining: Language Modelling

- Source task: Language modelling
- Cheap: no human annotations required
- Large amounts of publicly available data: Wikipedia, Web etc.

# Pretraining: Language Modelling

- Source task: Language modelling
- Cheap: no human annotations required
- Large amounts of publicly available data: Wikipedia, Web etc.
- Learn both word and sentence representations

## Pretraining: Language Modelling

- Source task: Language modelling
- Cheap: no human annotations required
- Large amounts of publicly available data: Wikipedia, Web etc.
- Learn both word and sentence representations
- Many variants in objective functions:

# Pretraining: Language Modelling

- Source task: Language modelling
- Cheap: no human annotations required
- Large amounts of publicly available data: Wikipedia, Web etc.
- Learn both word and sentence representations
- Many variants in objective functions:
  - predict all context words given target word (and vice versa)

# Pretraining: Language Modelling

- Source task: Language modelling
- Cheap: no human annotations required
- Large amounts of publicly available data: Wikipedia, Web etc.
- Learn both word and sentence representations
- Many variants in objective functions:
  - predict all context words given target word (and vice versa)
  - predict masked context words (fill-in-the-blank)

# Pretraining: Language Modelling

- Source task: Language modelling
- Cheap: no human annotations required
- Large amounts of publicly available data: Wikipedia, Web etc.
- Learn both word and sentence representations
- Many variants in objective functions:
  - predict all context words given target word (and vice versa)
  - predict masked context words (fill-in-the-blank)
  - predict perturbed context

# Pretraining: Language Modelling

- Source task: Language modelling
- Cheap: no human annotations required
- Large amounts of publicly available data: Wikipedia, Web etc.
- Learn both word and sentence representations
- Many variants in objective functions:
  - predict all context words given target word (and vice versa)
  - predict masked context words (fill-in-the-blank)
  - predict perturbed context
- Models?

# Pretraining: Models

- Feedforward networks, e.g. word2vec<sup>3</sup>

---

<sup>3</sup>Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119)

<sup>4</sup>Howard, J., & Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (pp. 328-339).

<sup>5</sup>Vaswani, Ashish, et al. (2017) "Attention is all you need." Advances in neural information processing systems.

<sup>6</sup>Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies(pp. 4171-4186).

# Pretraining: Models

- Feedforward networks, e.g. word2vec<sup>3</sup>
- LSTM, e.g. Universal Language Model Fine-tuning (ULMFiT<sup>4</sup>)

---

<sup>3</sup>Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119)

<sup>4</sup>Howard, J., & Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (pp. 328-339).

<sup>5</sup>Vaswani, Ashish, et al. (2017) "Attention is all you need." Advances in neural information processing systems.

<sup>6</sup>Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies(pp. 4171-4186).

# Pretraining: Models

- Feedforward networks, e.g. word2vec<sup>3</sup>
- LSTM, e.g. Universal Language Model Fine-tuning (ULMFiT<sup>4</sup>)
- Transformer<sup>5</sup> Network, e.g. Bidirectional Encoder Representations from Transformers (BERT<sup>6</sup>)

---

<sup>3</sup>Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119)

<sup>4</sup>Howard, J., & Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (pp. 328-339).

<sup>5</sup>Vaswani, Ashish, et al. (2017) "Attention is all you need." Advances in neural information processing systems.

<sup>6</sup>Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies(pp. 4171-4186).

# BERT: Pre-training of Deep Bidirectional Transformers

- Encoder 6 layers: 2 sub-layers each

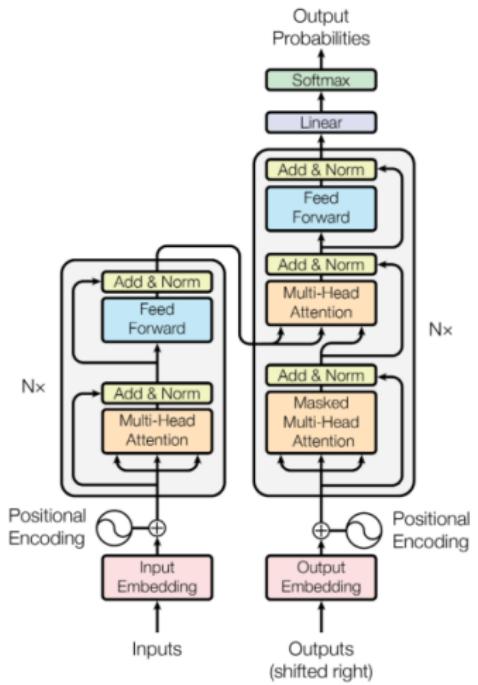
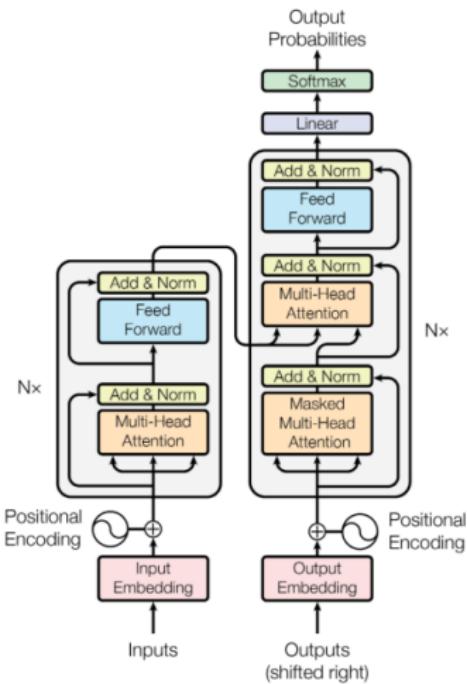


Figure 1: The Transformer - model architecture.

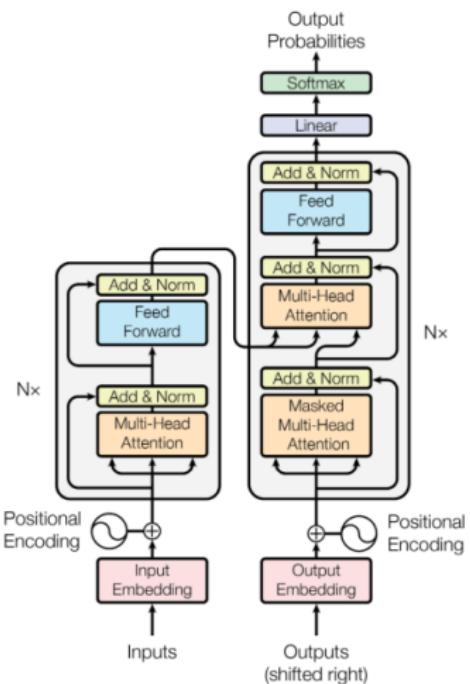
# BERT: Pre-training of Deep Bidirectional Transformers



- Encoder 6 layers: 2 sub-layers each
- Sub-layer 1: Multi-head self-attention mechanism (Lectures 7-9)

Figure 1: The Transformer - model architecture.

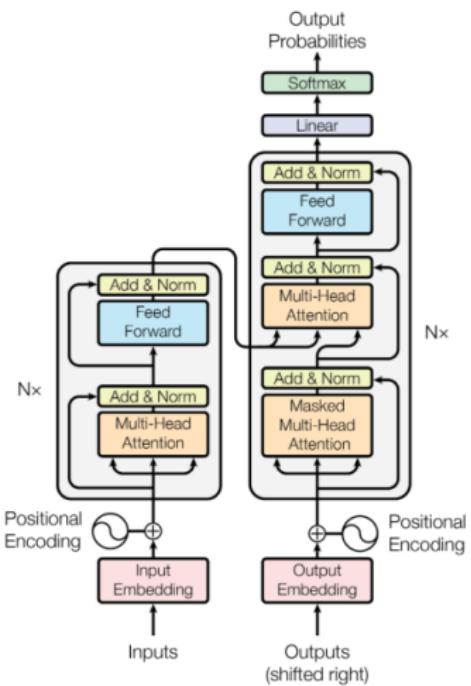
# BERT: Pre-training of Deep Bidirectional Transformers



- Encoder 6 layers: 2 sub-layers each
- Sub-layer 1: Multi-head self-attention mechanism (Lectures 7-9)
- Sub-layer 2: Position-wise fully connected layer

Figure 1: The Transformer - model architecture.

# BERT: Pre-training of Deep Bidirectional Transformers



- Encoder 6 layers: 2 sub-layers each
- Sub-layer 1: Multi-head self-attention mechanism (Lectures 7-9)
- Sub-layer 2: Position-wise fully connected layer
- Output of each sublayer is combined with its input followed by layer norm

Figure 1: The Transformer - model architecture.

# BERT: Pre-training of Deep Bidirectional Transformers

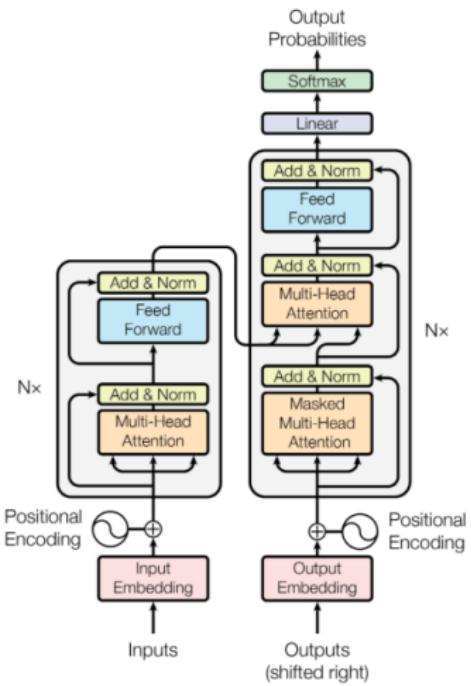
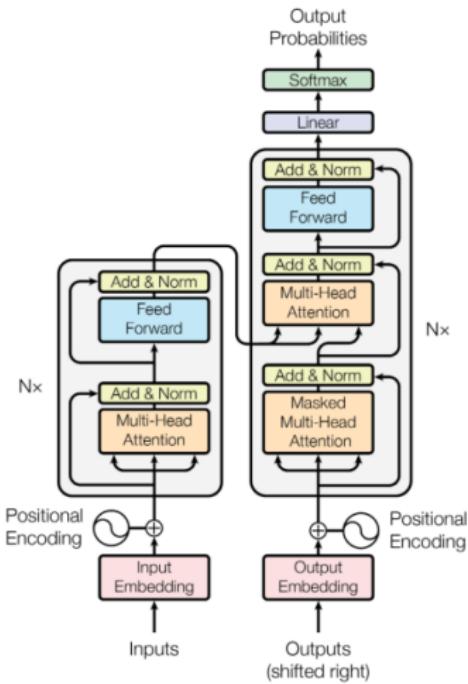


Figure 1: The Transformer - model architecture.

- Encoder 6 layers: 2 sub-layers each
- Sub-layer 1: Multi-head self-attention mechanism (Lectures 7-9)
- Sub-layer 2: Position-wise fully connected layer
- Output of each sublayer is combined with its input followed by layer norm
- Input is combined with a positional embedding (containing information for particular position in the sequence)

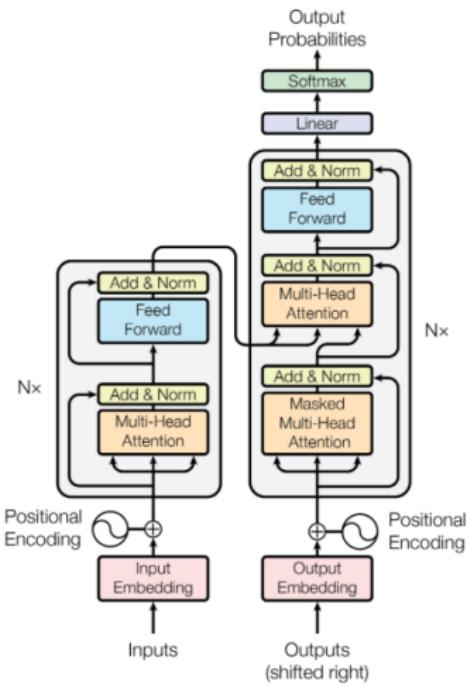
# BERT: Pre-training of Deep Bidirectional Transformers



- Decoder 6 layers: 3 sub-layers each

Figure 1: The Transformer - model architecture.

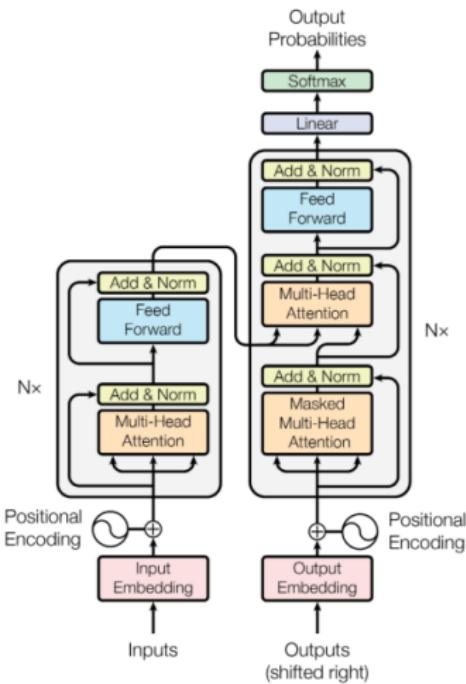
# BERT: Pre-training of Deep Bidirectional Transformers



- Decoder 6 layers: 3 sub-layers each
- Sub-layer 3: multihead attention over the output of the encoder

Figure 1: The Transformer - model architecture.

# BERT: Pre-training of Deep Bidirectional Transformers



- Decoder 6 layers: 3 sub-layers each
- Sub-layer 3: multihead attention over the output of the encoder
- Output layer

Figure 1: The Transformer - model architecture.

# BERT: Pre-training of Deep Bidirectional Transformers

- BERT is a bidirectional Transformer

# BERT: Pre-training of Deep Bidirectional Transformers

- BERT is a bidirectional Transformer
- Pretrained on two tasks:

# BERT: Pre-training of Deep Bidirectional Transformers

- BERT is a bidirectional Transformer
- Pretrained on two tasks:
  - Masked LM

# BERT: Pre-training of Deep Bidirectional Transformers

- BERT is a bidirectional Transformer
- Pretrained on two tasks:
  - Masked LM
  - Next sentence prediction

# BERT: Pre-training of Deep Bidirectional Transformers

- BERT is a bidirectional Transformer
- Pretrained on two tasks:
  - Masked LM
  - Next sentence prediction
- Input:

# BERT: Pre-training of Deep Bidirectional Transformers

- BERT is a bidirectional Transformer
- Pretrained on two tasks:
  - Masked LM
  - Next sentence prediction
- Input:
  - WordPiece embeddings

# BERT: Pre-training of Deep Bidirectional Transformers

- BERT is a bidirectional Transformer
- Pretrained on two tasks:
  - Masked LM
  - Next sentence prediction
- Input:
  - WordPiece embeddings
  - [CLS] (special classification token) is appended at the beginning of each sequence

# BERT: Pre-training of Deep Bidirectional Transformers

- BERT is a bidirectional Transformer
- Pretrained on two tasks:
  - Masked LM
  - Next sentence prediction
- Input:
  - WordPiece embeddings
  - [CLS] (special classification token) is appended at the beginning of each sequence
  - Sentences are separated with a special token [SEP]

# BERT: Pre-training of Deep Bidirectional Transformers

- BERT is a bidirectional Transformer
- Pretrained on two tasks:
  - Masked LM
  - Next sentence prediction
- Input:
  - WordPiece embeddings
  - [CLS] (special classification token) is appended at the beginning of each sequence
  - Sentences are separated with a special token [SEP]
  - Token input representation: summing token, segmentation and position embeddings

# BERT: Pre-training of Deep Bidirectional Transformers

- BERT is a bidirectional Transformer
- Pretrained on two tasks:
  - Masked LM
  - Next sentence prediction
- Input:
  - WordPiece embeddings
  - [CLS] (special classification token) is appended at the beginning of each sequence
  - Sentences are separated with a special token [SEP]
  - Token input representation: summing token, segmentation and position embeddings
- The final hidden state corresponding to [CLS] token is used as the aggregate sequence representation for classification tasks (e.g. target tasks)

# BERT: Pre-training of Deep Bidirectional Transformers

- BERT is a bidirectional Transformer
- Pretrained on two tasks:
  - Masked LM
  - Next sentence prediction
- Input:
  - WordPiece embeddings
  - [CLS] (special classification token) is appended at the beginning of each sequence
  - Sentences are separated with a special token [SEP]
  - Token input representation: summing token, segmentation and position embeddings
- The final hidden state corresponding to [CLS] token is used as the aggregate sequence representation for classification tasks (e.g. target tasks)
- BERT variants: XLNet, RoBERTa, ALBERT

# Adaptation

- Initialise your encoder on the target task using the weights you learned in LM

# Adaptation

- Initialise your encoder on the target task using the weights you learned in LM
- Change the output layer of your network to match the target task

# Adaptation

- Initialise your encoder on the target task using the weights you learned in LM
- Change the output layer of your network to match the target task
- Freeze the weights of the pretrained word embeddings/encoder

# Adaptation

- Initialise your encoder on the target task using the weights you learned in LM
- Change the output layer of your network to match the target task
- Freeze the weights of the pretrained word embeddings/encoder
- Learn the weights of the output layer on the target task data

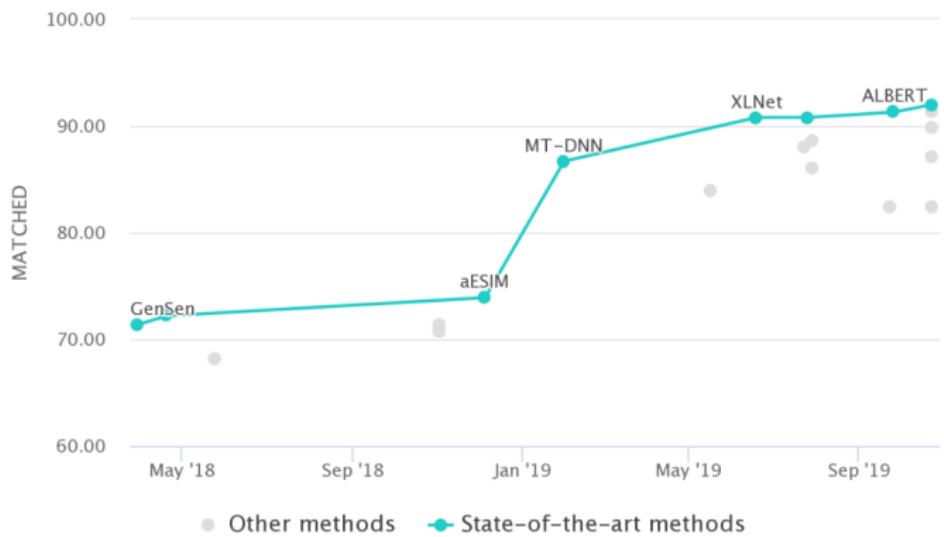
# Adaptation

- Initialise your encoder on the target task using the weights you learned in LM
- Change the output layer of your network to match the target task
- Freeze the weights of the pretrained word embeddings/encoder
- Learn the weights of the output layer on the target task data
- Unfreeze the weights of the pretrained components and fine-tune them (additional training steps with very small learning rate)

# Adaptation

- Initialise your encoder on the target task using the weights you learned in LM
- Change the output layer of your network to match the target task
- Freeze the weights of the pretrained word embeddings/encoder
- Learn the weights of the output layer on the target task data
- Unfreeze the weights of the pretrained components and fine-tune them (additional training steps with very small learning rate)
- In ULMFiT, the LM encoder (LSTM) is fine-tuned on the target task data before adaptation

# Does it work?



Performance on Natural Language Inference on MultiNLI<sup>7</sup>

<sup>7</sup><https://paperswithcode.com/sota/natural-language-inference-on-multinli>

# Bibliography

- [Blog post](#) on Transfer Learning by S. Ruder
- [Blog post](#) on Transfer Learning in NLP by S. Ruder
- [Blog post](#) on BERT by Samia

Thanks!