Practical Natural Language Processing

Julia Proskurnia
me@juliapro.xyz

July, 2019

About me, myself, I

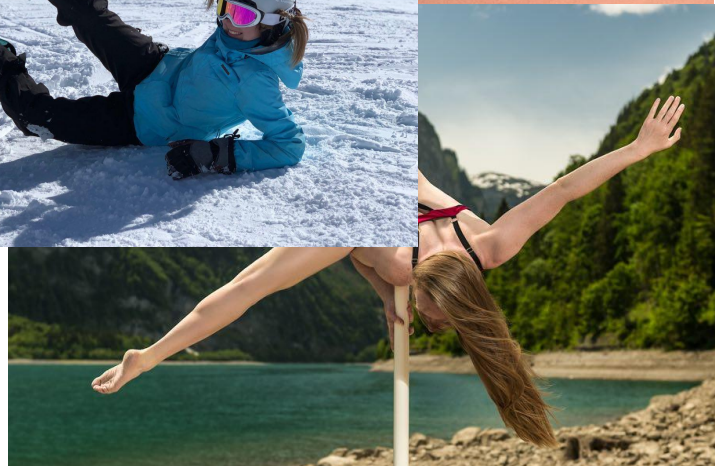
[BA, MS] NTUU KPI (Optimization, Neural Networks, Time Series Prediction)

[MS] EMDC, Barcelona, Stockholm (Distributed Systems)

[PhD] EPFL, Lausanne (Profiling, Modeling and Facilitating Online Activism)

Google, Zurich (Gmail Smartcompose boilerplate Personalization, Gmail Subject Smartcompose, et

Google, Zurich (Next Generation Assistant)



Course Expectations?

- Would be more useful for people less familiar with NLP.

- Familiarity with Machine Learning concepts would help, but we will cover that briefly as well.

What this course is not about!

- Going over some fancy math for FEW particular algorithms
- Coding scalable solutions
- Go into the wild and write DNN architectures left and right :)
- Not solely Deep Learning in NLP

But…

- You would get some practical starting points on many of the NLP directions
- Lots of reference points
- Buzzwords :)

Overview for what to expect

- Intro to NLP problems
- Intro to the course setup
- Text preprocessing and annotations
- N-Gram extraction, TF-IDF
- Document classification
- Word Embeddings
- Document classification with embeddings
- Topic modeling, clustering
- Intro to entity extraction
- Intro to sequence modeling, deep learning (CNN, LSTM)

Introduction to NLP problems

Structure of the Introduction

- What is Natural Language
- Examples, applications, a few demos
- Typical NLP tasks
- Evaluation and metrics
- The source of data or data collection

Natural Language Processing (or NLP)

Main objective:

program **computers** to **process and analyze** **large amount** of **natural language** data.

What is Natural Language?

"Natural" because evolved naturally, in contrast with Formal Languages.

Features of natural language:

1. Use of shared knowledge
   - Time ***flies like*** an arrow.
   - Play ***rolling stones*** on youtube
2. Concise, but ambiguous
   - Remove the stones from the cherries and put ***them*** in the pie.
   - The hunter shot the tiger; ***his*** wife too.
3. Unlimited expressive power
   - She was eating a fish ***with*** (bones, anger, some friends, a fork)
   - Congratulations! Great job!

Challenges for NLP

- Implicit -> allows conciseness (but potentially ambiguous)
- Unlimited expressive power -> flexible interpretation rules
  - (meanings != word as it is written)
- Why do we still understand each other? -> A lot of shared knowledge!

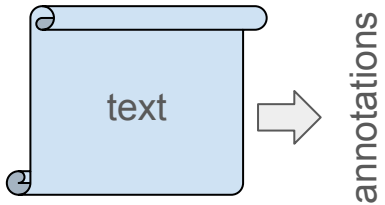
Moreover...

- Existing words (or linguistic units) change over time.
- New words are added.
- Language contains errors.
- Linguistic units denote an idea or even several.
- We have lots of common sense knowledge and understand hierarchical structures in language.

Challenges for NLP

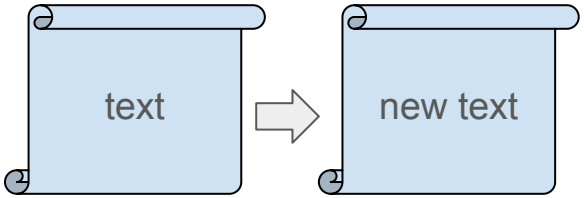
- Full understanding of Natural Language is difficult
  - Similar to understanding / modeling the full world knowledge, needs full-capacity AI
- Natural languages is highly dimensional and sparse
  - There are millions of words and most of them are very rare
- Internationalization requires prior within-domain knowledge
  - Many languages with scarce resources

Text to annotations



annotations	problem
a label (e.g. spam / not spam)	classification
a numerical value (e.g. positive strength)	regression
structured annotation (e.g. a syntax tree)	parsing
label sequence (per words, e.g. part-of-speech tags)	sequence labeling

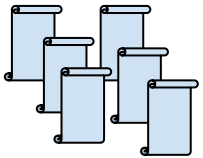
Text to another text



new text	problem
from one language to another	translation
from question to answer	question-answering
from text to its shortened version	text summarization
text continuation (e.g. SmartCompose)	language modeling
generating text rewrites	paraphrase generation

Text collection to useful info

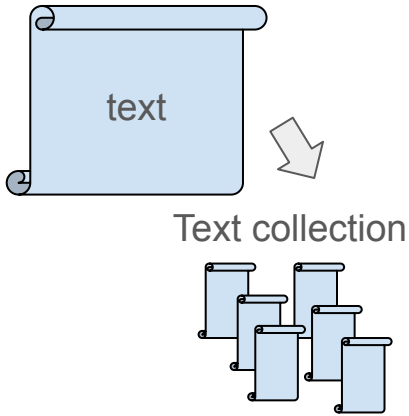
Text collection



useful info

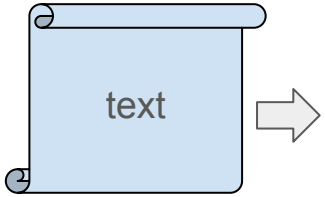
info	problem
find most representative texts, generating summary	summarization
identify facts / events (e.g. 'X is a capital of Y')	information extraction
find groups of similar texts (e.g. for deduplication)	clustering
identify different patterns (e.g. temporal or similarity-wise)	visualization

Text to text collection



	output	problem
text →	all relevant documents (e.g. search results for a query)	retrieval
text → Text collection	select a top doc from a collection (e.g. knowledge panel)	ranking

Space of NLP problems (non-exhaustive!)



annotations

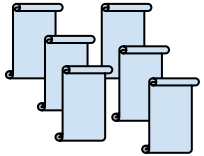
classification

regression

parsing

sequence labeling

Text collection



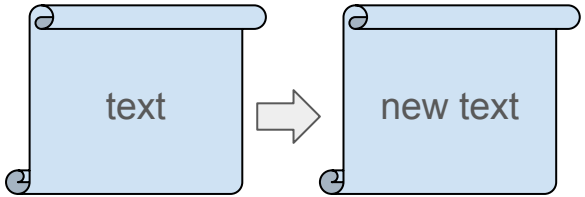
useful info

summarization

information extraction

clustering

visualization



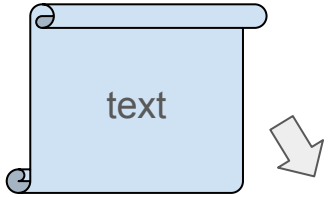
translation

question-answering

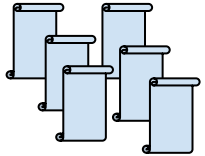
text summarization

language modeling

paraphrase generation



Text collection



retrieval

ranking

Applications and Examples

More examples in the colab...

Typical NLP tasks

- *Classification*: Classifying documents into particular categories (giving labels).
- *Regression*: Predict numerical values
- *Clustering*: Separating documents into non- overlapping subsets.
- *Ranking*: For a given input, rank documents according to some metric.
- *Association rule mining*: Infer likely associations patterns in data.
- *Structured output*: Parse trees, etc.
- *Sequence-to-sequence*: For a given input/source text, generate output annotations or another text.

Typical NLP Solution Flow

Example: Machine Translation

1	Formulate a problem statement	sequence-to-sequence generation
2	Collect the data	within-language texts, aligned text pairs
3	Get 'golden' or 'silver' answers	**golden:** human-rated or generated, **silver:** automatically aligned from Wikipedia
4	Search for prior resources	pre-trained text representations, ready tools for text annotation
5	Train the model	(many options)
6	Evaluate the quality	accuracy, readability, ...
7	Iterate	find better representations, more features, better data, etc.

Data sources

Raw text data (Unlabeled/Non-annotated):

- Pay for it :)
- Crawl it from the Web.
  - Examples: Scrappy, wiki/google crawler
  - IP addresses, proxies, throttling rates etc.
- Twitter (historical or livestream data (< 1% is free))
  - 1% historical tweets from archive.org.
  - 1% via Twitter API.
  - Keywords (up to 2K) that would match tweets in real time 1% total stream of messages.
- News media (News Archive, GDELT)
- Wikipedia (Wikipedia dumps)

Annotation of the data and pitfalls

- Annotate and/or even generate your data using CrowdSourcing
  a. Crowdflower
  b. MechanicalTurk, etc.
- Annotate using auxiliary lexical resources
  a. LIWC a tool that analyses your textual input on the presence of various
     i. "shades" - informal speech; syntactic structures; affect, social words; cognitive, perpetual, biological processes; relativity; personal concerns, etc.

Biases and pitfalls

- Biases
- Not representative
- Noisy data
- Incomplete data
- Incorrect data
- Missing data, etc.

Course practicum setup

- We will use colab.research.com.
- We will focus on the open-source libraries for NLP to give you easier explanations.

Actions:

- Either
  - Open colabs etc.
  - OR open the copy of the colab with jupyter notebook [RECOMMENDED if you want to try out dependency parsing]

Preprocessing and data cleaning

Structure of the Preprocessing and Data Cleaning

- Tokenization + Sentence Splitting
- Normalization / Text cleaning
  - Stemming/lemmatization, stop-words, punctuations, emojis, domain specifics, etc.
- Text Annotation
  - Part-Of-Speech Tagging
  - Parse Trees

Data description: Tweets about natural disasters

DataFrame name: **course_data**

- ‘label’ - whether the tweet is relevant to natural disasters or not
- ‘text’ - raw tweet strings
- ‘text_tokenized’ tokenized input texts
- ‘text_tokenized_cleaned’ tokenized and normalized text

document_embeddings - Text representation with word embedding

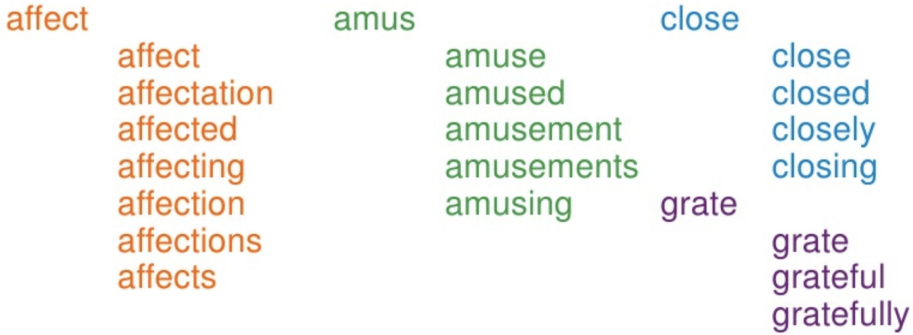
document_term_matrix - Bag-of-words representation

corpus_tokens - List with list of cleaned tokens of each document

POS exercise in colab

Stemming and Lemmatization

- **Stemming** is typically faster, more naive word shortening. (crude affix chopping)
  It is language dependent.



© Lightweight NLP

- **Lemmatization** take deeper approach to extracting lemmas of the words (part of speech, language specific dictionary, etc.).
  Lemmatization implies doing “proper” reduction to dictionary headword form
  am, are, is → be
  car, cars, car's, cars' → car

Porter Stemmer

Step 1a

sses	→ ss	caresses	→ caress
ies	→ i	ponies	→ poni
ss	→ ss	caress	→ caress
s	→ ø	cats	→ cat

Step 1b

(*v*)ing	→ ø	walking	→ walk
		sing	→ sing
(*v*)ed	→ ø	plastered	→ plaster
…			

Step 2 (for long stems)

ational	→ ate	relational	→ relate
izer	→ ize	digitizer	→ digitize
ator	→ ate	operator	→ operate
…			

Step 3 (for longer stems)

al	→ ø	revival	→ reviv
able	→ ø	adjustable	→ adjust
ate	→ ø	activate	→ activ
…			

Sample stemmings

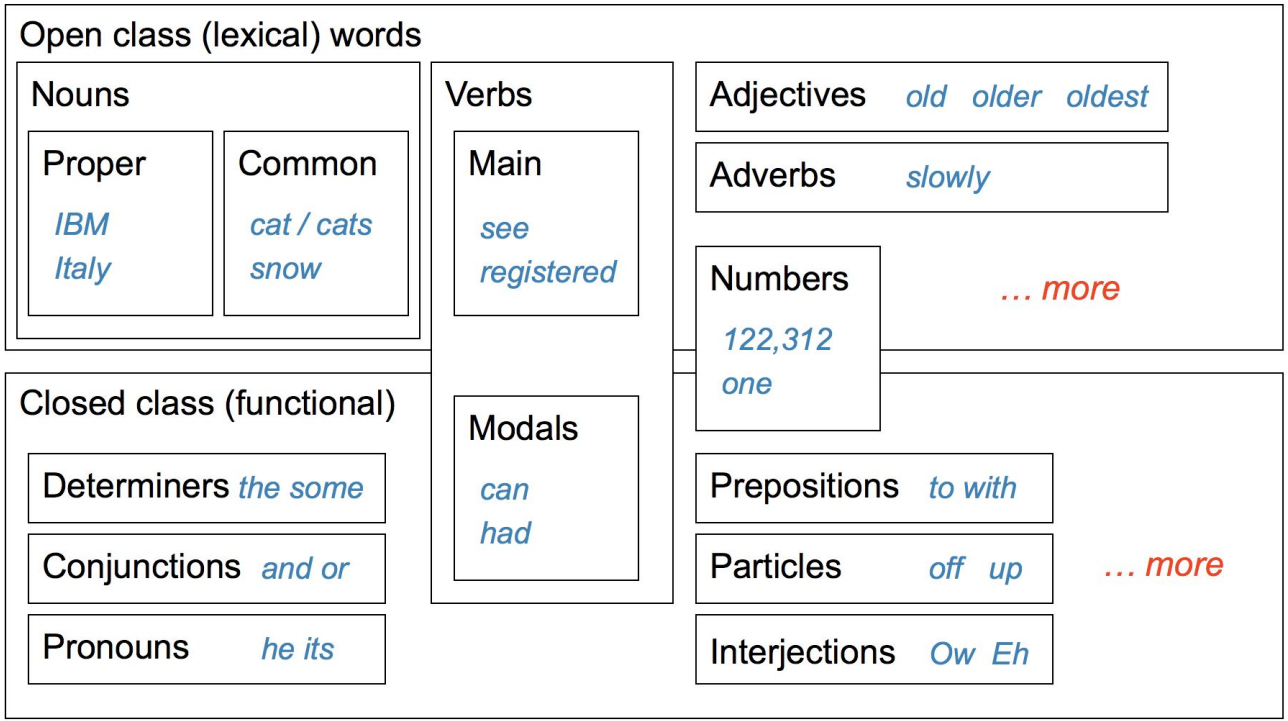
Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

Part-of-Speech Tagging



Part-of-Speech

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	*+,%, &*
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	*$*
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	*#*
PDT	Predeterminer	*all, both*	"	Left quote	' or "
POS	Possessive ending	*'s*	"	Right quote	' or "
PRP	Personal pronoun	*I, you, he*	(	Left parenthesis	[, (, {, <
PRP$	Possessive pronoun	*your, one's*	)	Right parenthesis	],), }, >
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	. ! ?
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	: ; ... – -
RP	Particle	*up, off*			

POS Tagging

- Words often have more than one POS: *back*
  - The *back* door = JJ
  - On my *back* = NN
  - Win the voters *back* = RB
  - Promised to *back* the bill = VB
- The POS tagging problem is to determine the POS tag for a particular instance of a word.

  - I know *that* he is honest = IN
  - Yes, *that* play was nice = DT
  - You can't go *that* far = RB

  40% of the word tokens are ambiguous

POS Tagging

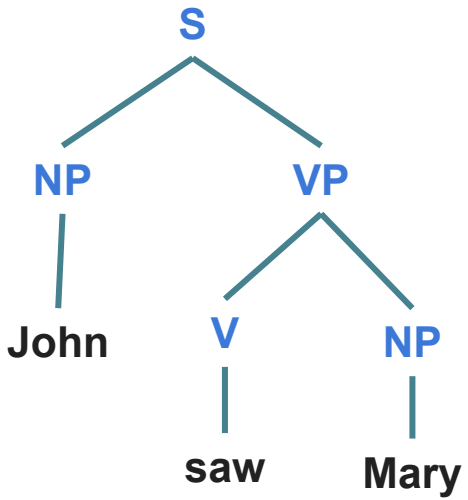
- What are the main sources of information for POS tagging?
  - Knowledge of neighboring words
    - Bill saw that man yesterday
    - NNP NN DT NN NN
    - VB VB(D) IN VB NN
  - Knowledge of word probabilities
    - *man* is rarely used as a verb….

• Word	the: the → DT
• Lowercased word	Importantly: importantly → RB
• Prefixes	unfathomable: un- → JJ
• Suffixes	Importantly: -ly → RB
• Capitalization	Meridian: CAP → NNP
• Word shapes	35-year: d-x → JJ

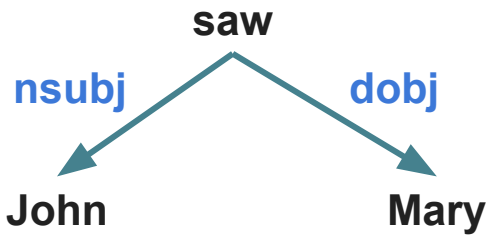
- The latter proves the most useful, but the former also helps

Parse Trees

Constituency trees:
structure of nested phrases/constituents



Dependency trees:
actual dependencies between words



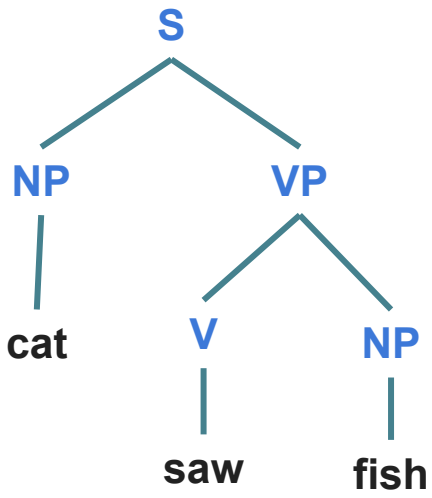
Parse Trees: Why to use?

- To find the root / main intent of the query / sentence.
- To find the modifiers / arguments of known frames (e.g. set alarm __WHEN__).
- Build event-like feature representations: nsubj('<someone>', 'saw').
- Extract smarter n-grams based on parse trees.

Constituency Parse Trees: Build with Probabilistic Context-Free Grammars

S	→ NP VP	1.0	
NP	→ NP NP	0.1	
NP	→ NP PP	0.3	
NP	→ N	0.6	
PP	→ P NP	1.0	
VP	→ V NP	0.7	
VP	→ VP PP	0.3	

N	→ cat	0.4	
N	→ fish	0.4	
N	→ saw	0.2	
V	→ meets	0.4	
V	→ saw	0.2	
V	→ greets	0.4	



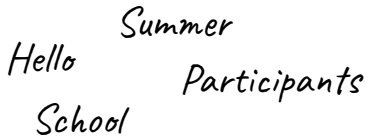
Text representation

Text representations

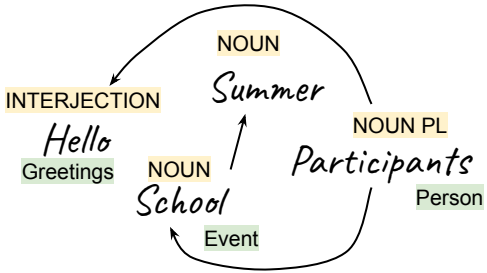
Idea: Represent the text so that the machine can perform its processing.
The text representations are usually inputs for machine-learning models.

Different approaches*:

Lexical

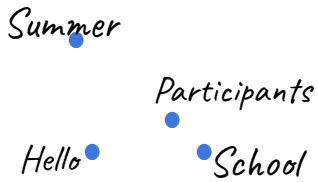


Linguistic



Entities

Embeddings



* (that we will discuss)

Bag-of-words (BOW) representation

Idea: Split the text into tokens (~words), extract their counts, and build their vector.

Input text: *"Hello, Summer school participants!"*

Set of normalized **words**: hello, summer, school, participants

As a **count vector**: [0 0 0 1 0 0 0 … 0 1 0 0 … 0 0 1 0 … 0 0 0 1 0 …0]

(hello) (summer) (participants) (school)

TF-IDF, Conditional Probability, etc.

Weighting tokens/features:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

P (word_i | document_j) / P (word_i | all_documents)

Removing noise:

- Remove tokens that are very frequent
- Remove tokens that have very low TF-IDF

Compare frequency distributions of 2 independent datasets and diminish the intersection.

Bag-of-words: Pros & Cons

Pros:

- Very simple, no need for special annotations

Cons:

- Can have too many low-occurring words
- Breaking collocations, e.g. "happy hour", "new york"
- Breaking syntax structure, e.g. "John saw Mary" or "Mary saw John"
- Purely lexical, no notion of semantics:
  - Different representations for similar concepts, e.g. synonyms ("phone" vs. "telephone")
  - Same representation of ambiguous words, e.g. "buy jaguar car" vs. "jaguar in nature"

N-grams representation

Idea: Take all longer consecutive token sequences as a vector.

Input text: *"Hello, Summer school participants!"*

Normalized text: "hello gwe school participants"

Extracted n-grams:

- unigrams: hello, summer, school, participants
- bigrams: hello summer, summer school, school participants
- trigrams: hello summer school, summer school participants
- 4-grams: hello summer school participants

N-grams: Pros & Cons

Pros:

- Can capture some structure and connotations
- Quite powerful

Cons:

- Aggravated data sparsity
  - Longer n-grams appear even less frequently than particular unigrams
- Correlated features
  - BUT: still works in practice
- Plus other problems for lexical-level representation

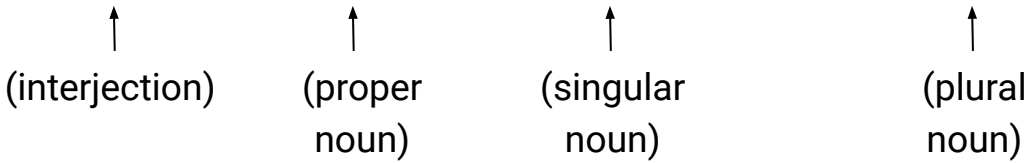
Linguistic representation: POS (Part-of-Speech)Tags

Idea: Capture better word meanings by assigning a POS tag to each word.

Why? Some words can have several POS tags: back (JJ/adjective) vs. back (NN/noun)

Input text: *"Hello, Summer school participants!"*

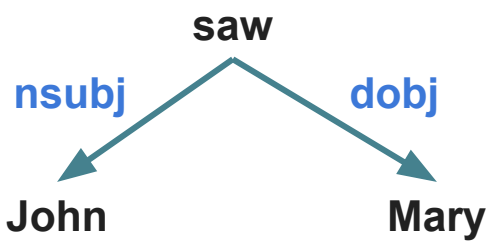
Annotated text: Hello/UH Summer/NNP schjool/NN participants/NNS



Linguistic representation: Syntax parse trees

Idea: Capture the syntactic structure of the sentence in a tree format.

Dependency trees:
actual dependencies between words



Why?

- Can identify sentence root
- Can extract smarter lexical features:
  - hello participants from
    "Hello Summer school participants"

Linguistic representation: Semantic role labels

Why? Get more semantic text understanding in order for machine to act on it.

Input query: "play Rolling Stones on youtube"

Role labels annotations: "play/ACTION rolling stones/ARTIST on youtube/APP"

Current state: Application-specific, usually focusing on particular label sets.

Linguistic representations: Pros & Cons

Pros:

- Can link to more formal language representation (e.g. language grammar)
- Captures more meaning (better disambiguation & connotation structure)

Cons:

- Each layer of annotations adds errors and computational complexity

Entity representation

Idea: Link the lexical forms of entities mentioned in the text to their actual identities (entities defined by non-ambiguous ids).

Examples:

"buy jaguar" → /wordnet/jaguar_car_brand

"jaguar in the nature" → /wordnet/jaguar_animal

Final text representation: a vector of detected entities.

Entity representation: Pros & Cons

Pros:

- Can disambiguate lexical forms into entities, e.g. "jaguar" in context
- Can match different lexical forms into the same entity, e.g. "phone" and "telephone"
- Has information about entity relations, e.g. "/knowledgebase/iphone" is produced by "/knowledgebase/apple"

Cons:

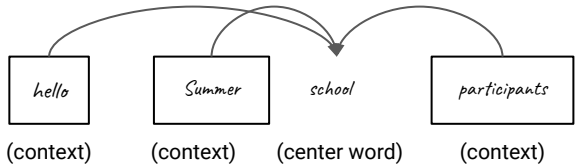
- Introduces the room for interpretation mistakes
- Only entities! Does not represent (yet) intents, common adjectives, nouns, etc.

Word embeddings: Dense semantic representation

Idea: Each word corresponds to a vector in multi-dimensional space.

Desired property:

Similar words are close to each other in that space, while unrelated words are further apart.



Sentence representation: Represent an input text as one multi-dimensional vector.

- Average embedding of all input words
- Train models to represent sequences (Deep Learning)

Word2vec visualization

Embeddings representation: Pros & Cons

Pros:

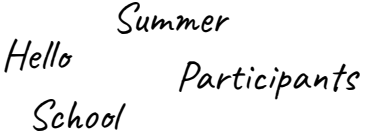
- Allow direct vector operations (dense representation)
- Can capture text semantics
- Really powerful at data generalizations

Cons:

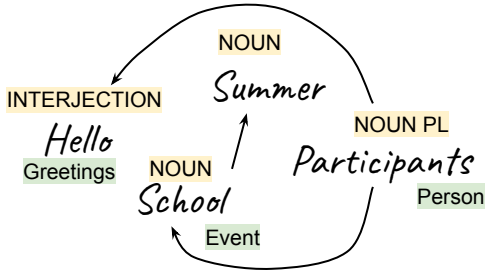
- Less interpretable
- Require tons of data for training
  - BUT: Pre-trained models are available!
- Computationally expensive
  - BUT: Often can be precomputed.

Text representations: Summary

Lexical

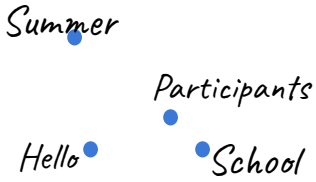


Linguistic



Entities

Embeddings

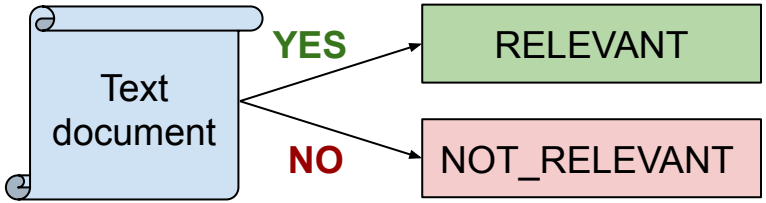


These representations are only the input for the models. The models can build even better representations of the full text by further processing them.

google/sentencepiece

Real text problems: Document classification

Document Classification

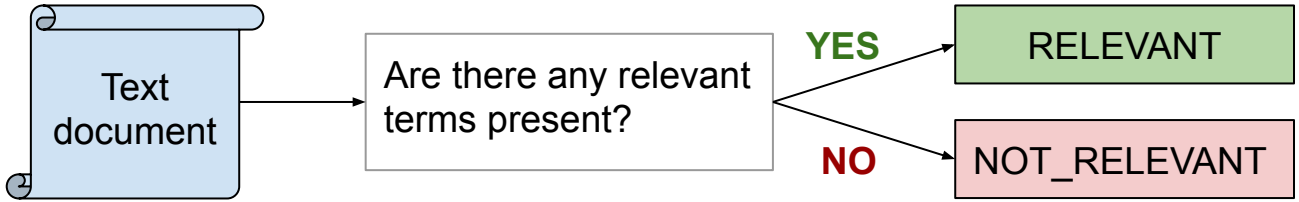


Given a set of labels/classes, need to assign labels to each document.

Examples:

- Spam filtering,
- sentiment analysis,
- documents retrieval (finding relevant documents),
- new topic classification

Rule-based classification



How to find which terms/words are relevant and which are not?

Building lexicons

Pointwise Mutual Information (PMI)

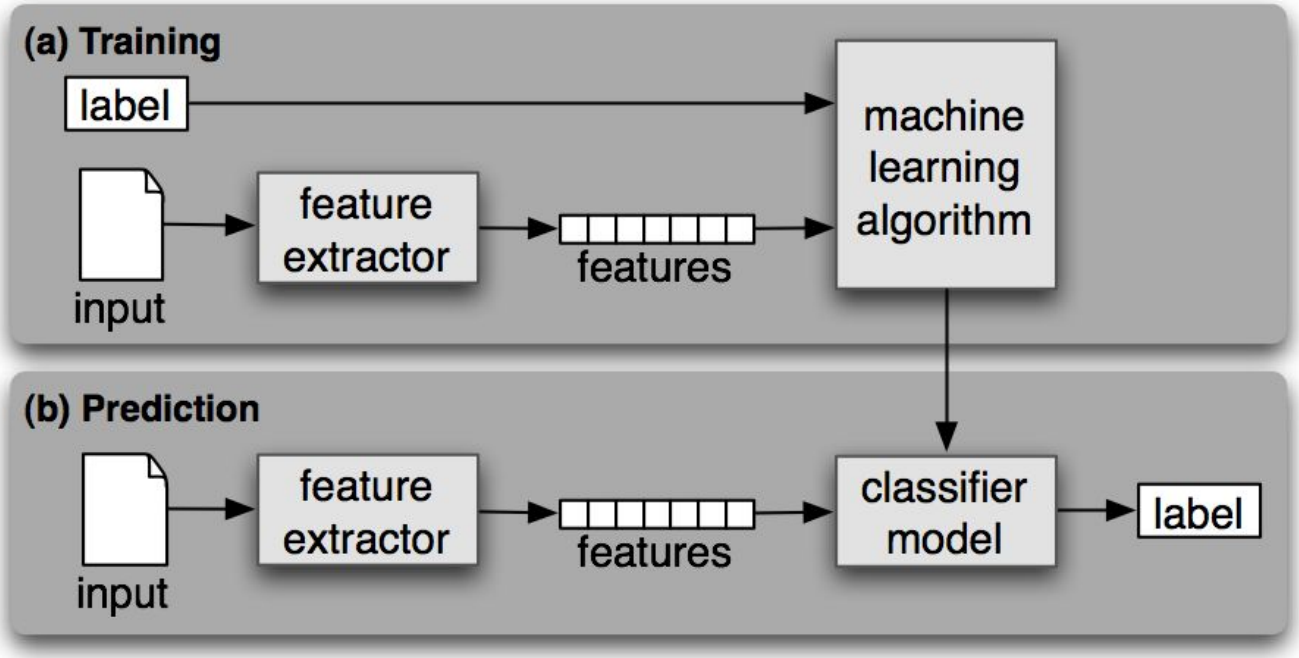
$$PMI(t, c) = \log_2 \frac{P(t,c)}{P(t)P(c)} = \log_2 \frac{P(t|c)}{P(t)}$$

term class

$$relatedness_{PMI}(t) = PMI(t, related) - PMI(t, not\ related) =$$

Document Classification

Supervised document classification



Supervised document classification: Common models

Generative models

- Try to generate the observed data from the hidden structure (labels y)
- Model joint probability of $p(X, y)$ and try to maximize joint likelihood
- Example: Naive Bayes

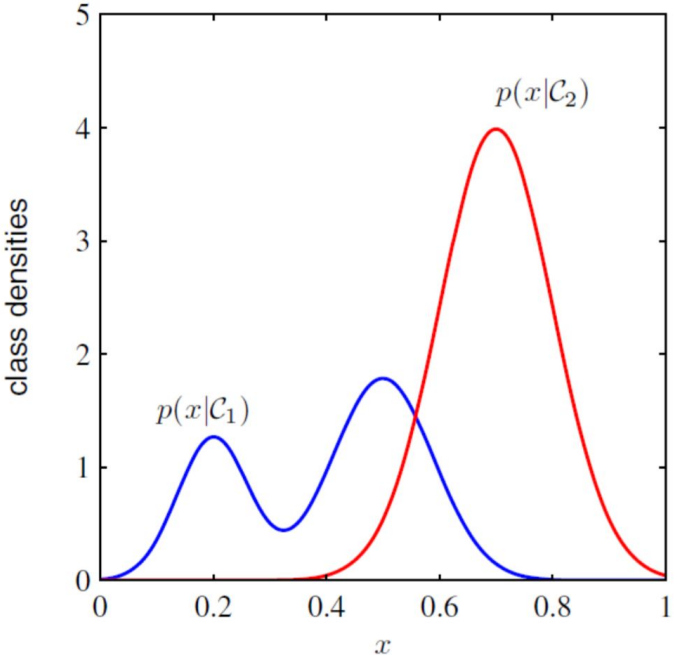
Discriminative models

- Directly estimate a decision rule/boundary
- Model conditional probability $p(y \mid X)$ and try to maximize conditional likelihood (give the data X)
- Example: Logistic Regression

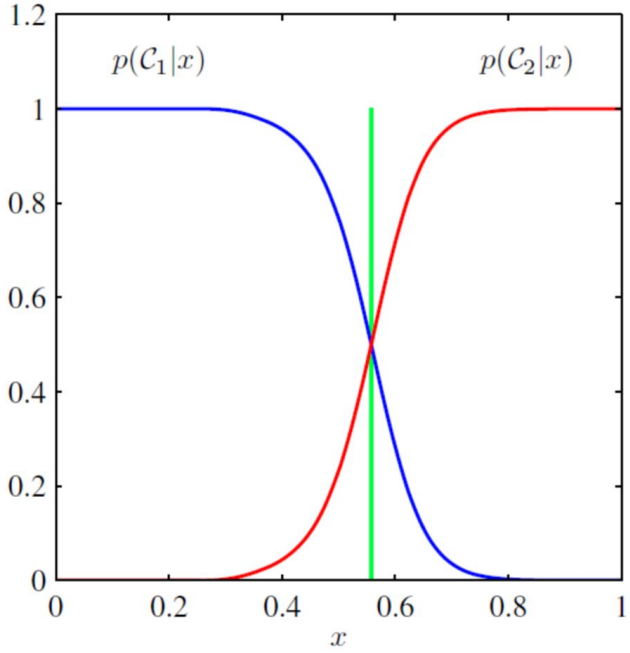
Generative vs. Discriminative

- Binary classification as an example

Generative Model's view



Discriminative Model's view



Naive Bayes

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(c \mid d)$$

MAP is "maximum a posteriori" = most likely class

$$= \underset{c \in C}{\operatorname{argmax}} \frac{P(d \mid c)P(c)}{P(d)}$$

Bayes Rule

$$= \underset{c \in C}{\operatorname{argmax}} P(d \mid c)P(c)$$

Dropping the denominator

Multinomial Naive Bayes

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(d \mid c) P(c)$$

$$= \underset{c \in C}{\operatorname{argmax}} P(x_1, x_2, \ldots, x_n \mid c) P(c)$$

Document d represented as features x1..xn

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c_j) \prod_{x \in X} P(x \mid c)$$

Multinomial Naive Bayes

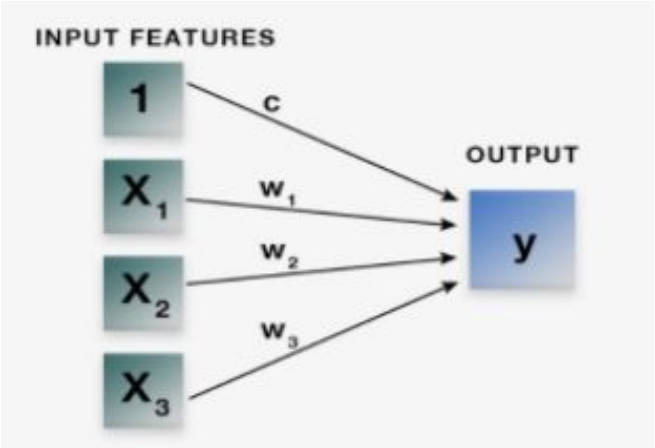
In such scenario, each document is assumed to be generated as follows.

1. Class y is generated according to $P(y)$.
2. X is generated by sequentially picking words from V (vocabulary) with replacement. Each word is picked with probability $P(w_j|y)$.

For example, the probability to generate a document $x = w_{j1} \dots w_{jL}$ is

$$P(x|y) = \prod_{l=1}^{L} P(w_{jl}|y) = \prod_{j=1}^{J} P(w_j|y)^{n_j(x)}$$

Linear Regression VS Logistic Regression



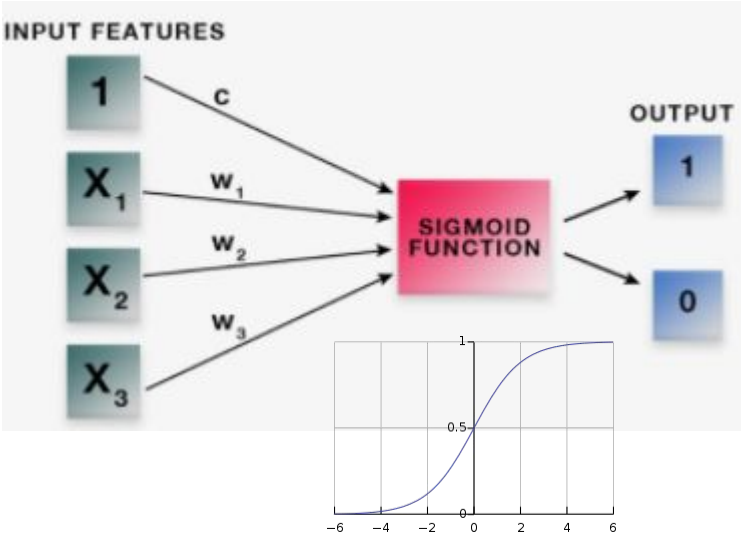


Image source: https://blog.quantinsti.com/machine-learning-logistic-regression-python/

Classification: Evaluation Metrics

	Actually correct	Actually incorrect
Selected by the algo	True positives	False positives
Not selected by the algo	False negatives	True negatives

Accuracy: % of items classified correctly = (tp + tn) / all

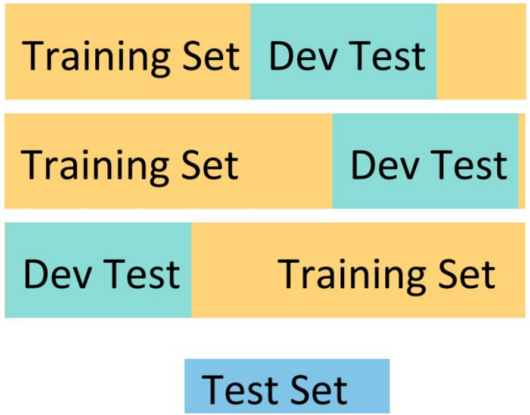
Precision: % of selected items that are correct = tp / (tp+fp)

Recall: % of correct items that are selected = tp / (tp + fn)

$$F = \frac{1}{\alpha \frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

Classification: Common evaluation setup

- Metric: P/R/F1 or Accuracy
- Unseen test set
  - avoid overfitting ('tuning to the test set')
  - more conservative estimate of performance
- Cross-validation over multiple splits
  - Handle sampling errors from different datasets
  - Pool results over each split
  - Compute pooled dev set performance



Typical approaches to classification

Type	Input	Idea
Rule-based	Explicit linguistic patterns, lexicons	Apply given rules
Supervised	High-quality annotated data	Learn machine-learning classifier based on annotated examples
Semi-supervised	Annotated & non-annotated data	Use more data to find better feature representation, learn more patterns, or mine new training examples
Distant supervision	Pseudo-annotated data based on some heuristics	Use heuristics to get more training data (often of lower quality or biased)

Text Representation: Word Embeddings

Word embeddings is a set of language modeling and feature learning techniques.

Key idea - dimensionality reduction that captures semantic/morphological/contextual/hierarchical/etc.

1. preserving the morphological structure (subword information, etc.);
2. word context representation;
3. global corpus statistics;
4. word hierarchy as in WordNet;
5. relationship between documents and the terms they contain.

Typical Word Represent with one-hot vectors

In traditional NLP, we regard words as discrete symbols:
hotel, conference, motel

Means one 1, the rest 0s

Words can be represented by one-hot vectors:

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

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

Drawbacks of the one-hot representation

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

But:

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

These two vectors are orthogonal.

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

Solution:

- Could rely on WordNet's list of synonyms to get similarity?

How can we introduce the meaning to the machine?

Common solution: Use e.g. WordNet, a resource containing lists of **synonym sets** and **hypernyms** ("is a" relationships).

e.g. synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
for synset in wn.synsets("good"):
    print "(%s)" % synset.pos(),
    print ", ".join([l.name() for l in synset.lemmas()])
```

```
(adj) full, good
(adj) estimable, good, honorable, respectable
(adj) beneficial, good
(adj) good, just, upright
(adj) adept, expert, good, practiced,
proficient, skillful
(adj) dear, good, near
(adj) good, right, ripe
…
(adv) well, good
(adv) thoroughly, soundly, good
(n) good, goodness
(n) commodity, trade good, good
```

e.g. hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

Potential issues with the ontology, knowledge base solutions

- Great as a resource but missing nuance
  - e.g. “proficient” is listed as a synonym for “good”.
    This is only correct in some contexts.
- Missing new meanings of words
  - e.g. wicked, badass, nifty, wizard, genius, ninja, bombest
  - Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt
- Hard to compute accurate word similarity →

Maybe represent words by their contexts?

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

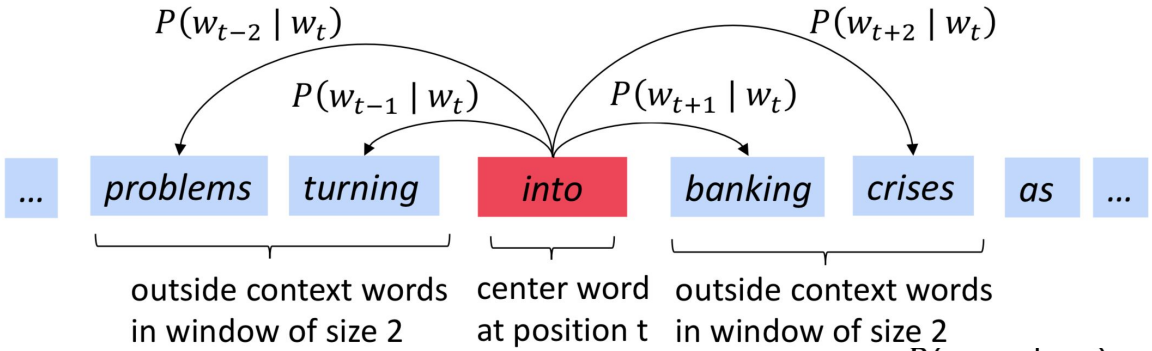
…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

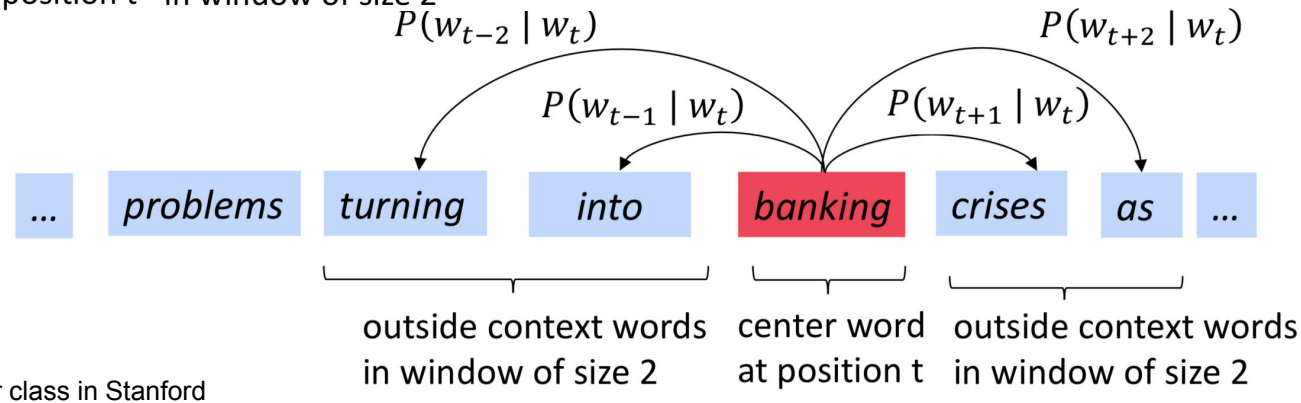
© Taken from the Richard Socher class in Stanford
http://cs224d.stanford.edu/lectures/CS224d-Lecture2.pdf

Word2Vec (Mikolov et al. 2015)

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

Given a center words compute P(...) of its context





© Taken from the Richard Socher class in Stanford
http://cs224d.stanford.edu/lectures/CS224d-Lecture2.pdf

Word2Vec: Objective Function

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

Likelihood =

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

θ is all variables to be optimized

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

sometimes called *cost* or *loss* function

minimize

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

Word2Vec: How to calculate p(neighbors | center)?

We will *use two* vectors per word *w*:

- v_w when *w* is a center word
- u_w when *w* is a context word

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

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

Dot product compares similarity of *o* and *c*.
Larger dot product = larger probability

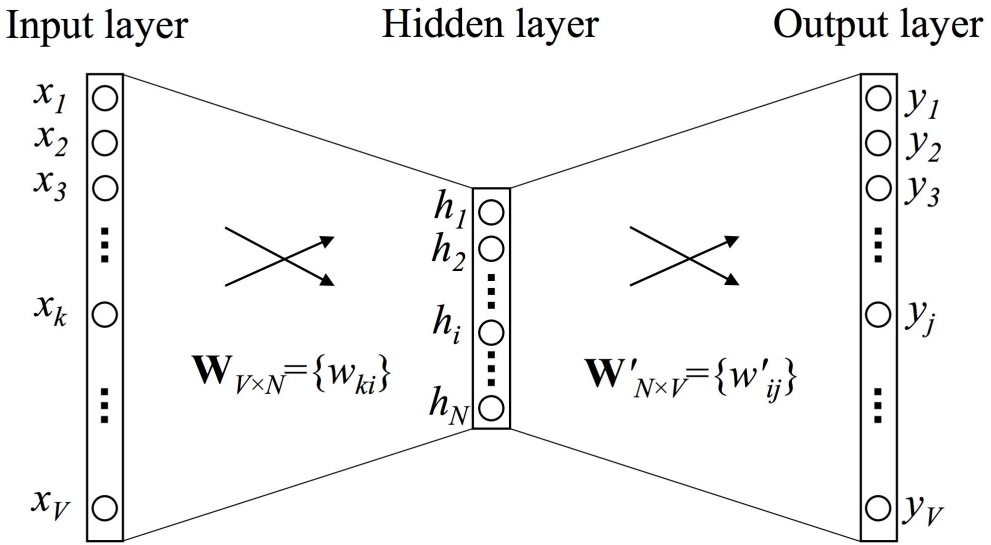
After taking exponent,
normalize over entire vocabulary

Word2Vec: Parameters of the model

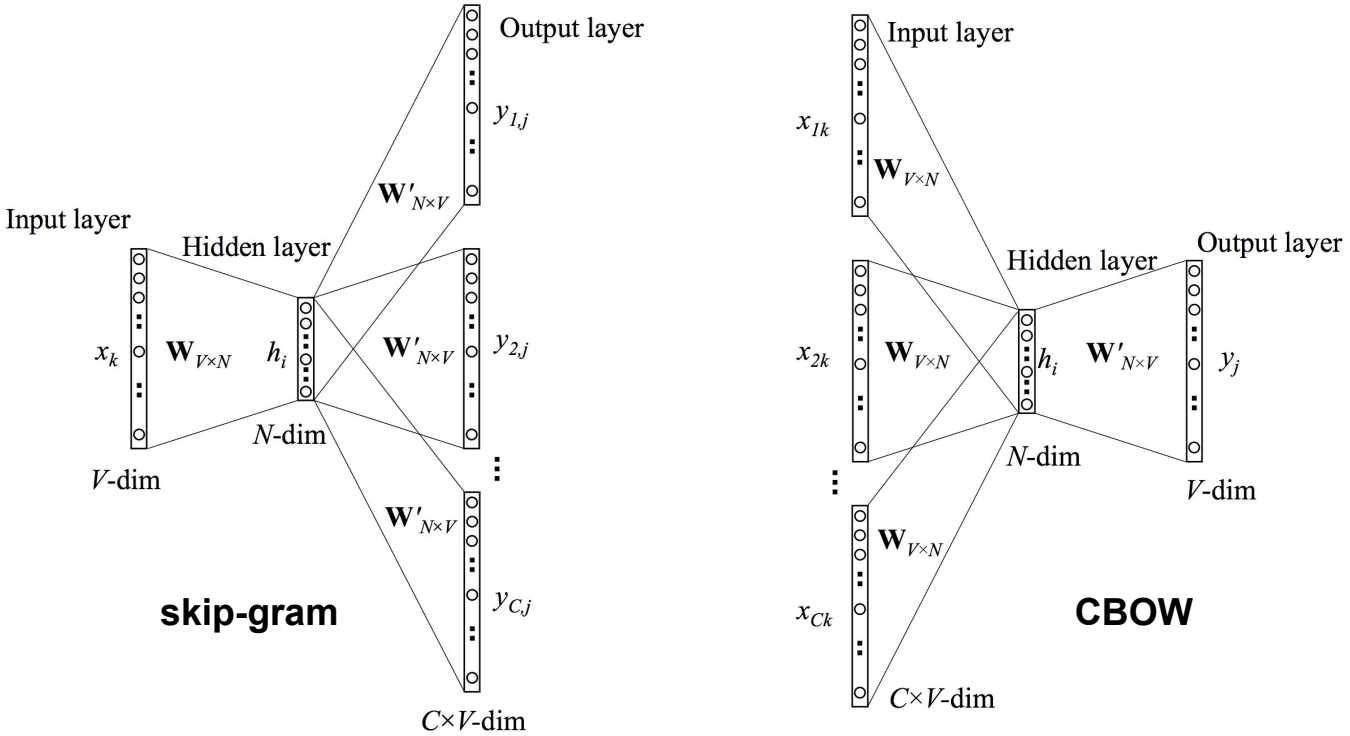
In our case with *d*-dimensional vectors and *V*-many words:

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

Word2vec One Context Word



Word Embeddings



© Taken from “word2vec Parameter Learning Explained”
https://arxiv.org/pdf/1411.2738.pdf

Word Embeddings Visualization

https://projector.tensorflow.org/

Document Similarity

Topic Modeling

Overview: Similarity, Topicality

Find a document that is more similar to a given input text?

- Similarity measures
  - Text based (edit distance)
  - Vector space representation based (euclidean distance, cosine similarity, WMD)

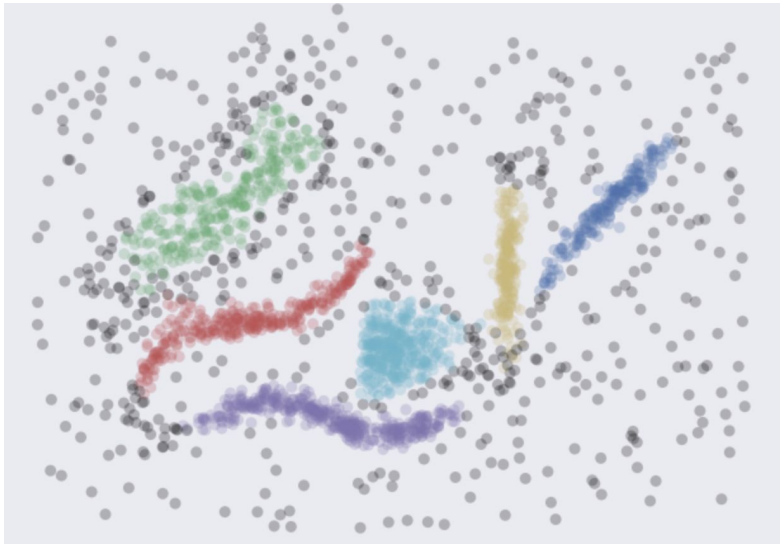
- Document similarity
  - Tf-idf representation
  - W2V representation

Clustering

Group input documents into known or unknown number of groups based on similarity between them.

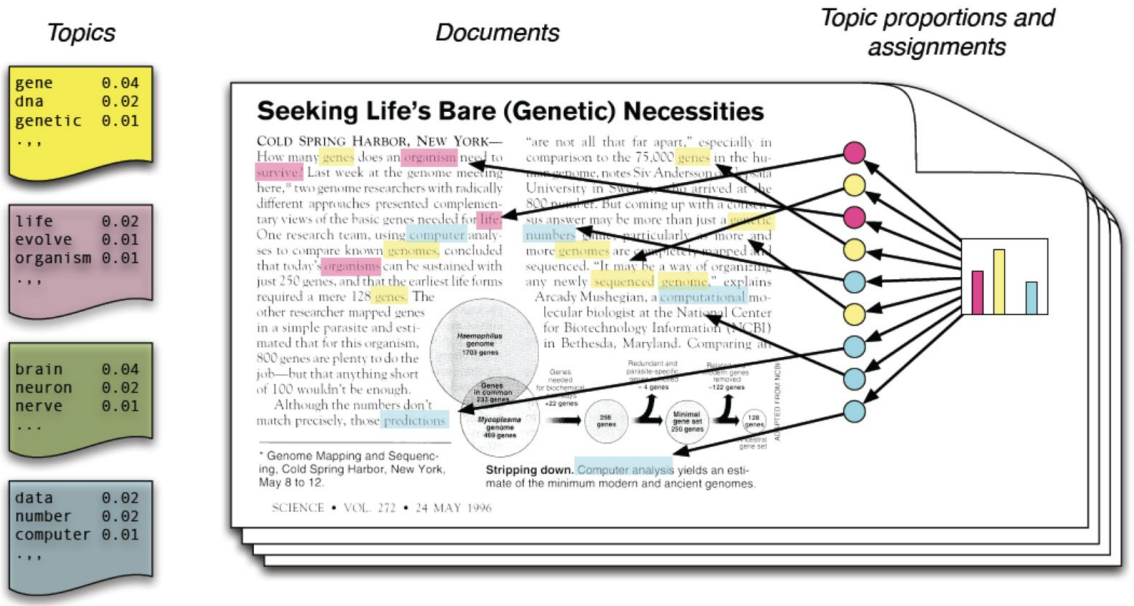
Why?

- Reduce input size
- Discover hidden patterns in your data without any supervision
- Visualize these patterns



© taken from lmcinnes github
http://nbviewer.jupyter.org/github/lmcinnes/hdbscan/blob/master/notebooks/Comparing%20Clustering%20Algorithms.ipynb

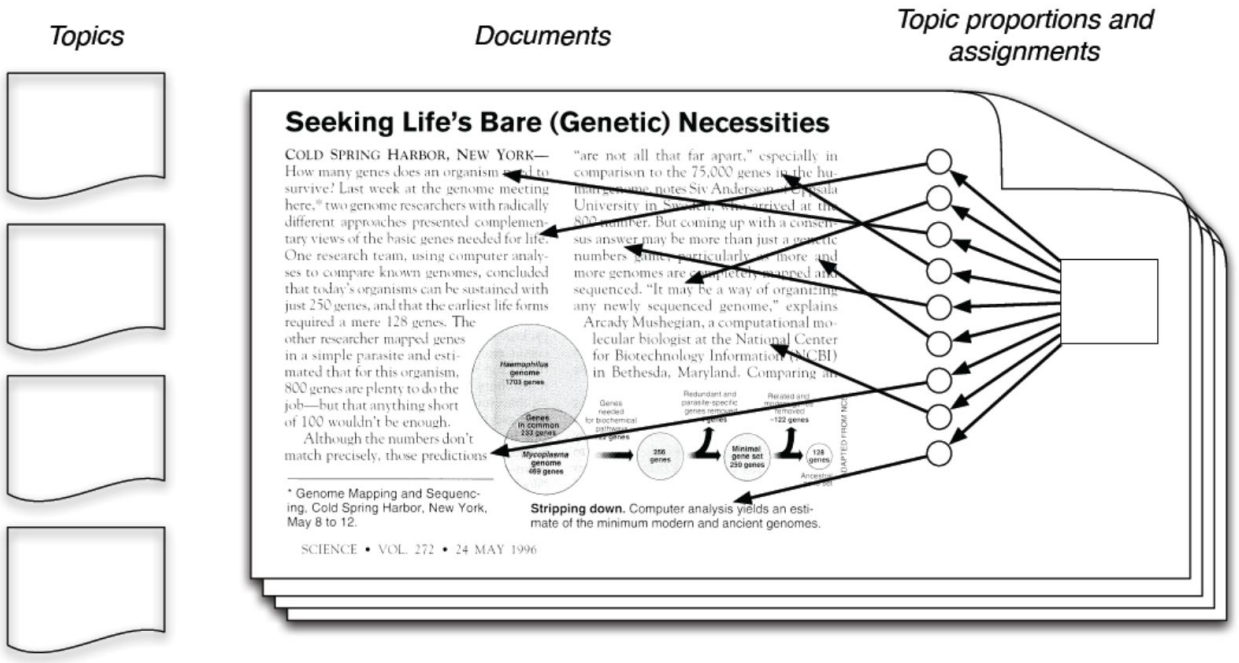
LDA



- Each **topic** is a distribution over words
- Each **document** is a mixture of corpus-wide topics
- Each **word** is drawn from one of those topics

© Whole set of slides here:
http://vision.stanford.edu/teaching/cs231b_spring1213/slides/mseg_topics.pdf

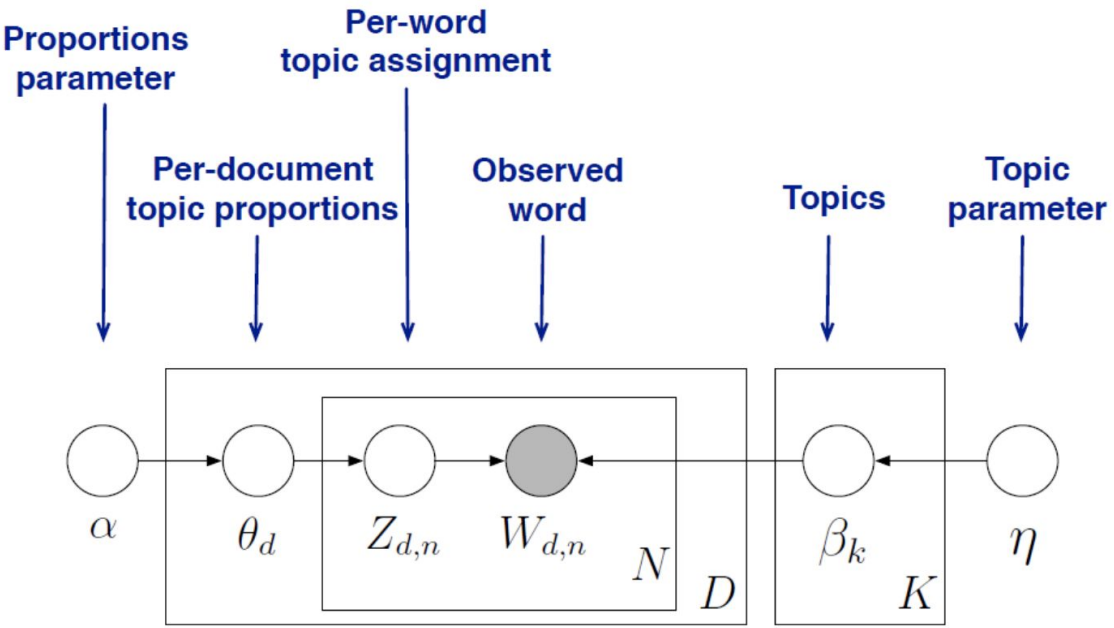
LDA



- Our goal is to **infer** the hidden variables
- I.e., compute their distribution conditioned on the documents

$$p(\text{topics}, \text{proportions}, \text{assignments} \mid \text{documents})$$

LDA



- Nodes are random variables; edges indicate dependence.
- Shaded nodes are observed; unshaded nodes are hidden.
- Plates indicate replicated variables.

General learning schema

To learn the topic representation of each document and the words associated to each topic:

- Go over the documents and **assign randomly** each word in the topic to one of the K topics
- for each document and each word in the document
  - for each topic t,
    (1) $p(topick|documentn)$ - the portion of words in document n that are currently assigned to the topic k;
    (2) $p(wordi|topick)$ - portion of assignments to topic k over all documents that come from this word i.
  - Reassign word i to a new topic, where we choose a topic with probability $p(topick|documentn) \cdot p(wordn|topick)$ - the probability that topic k generated word i.
- Stop once steady

Advanced NLP 1:
Linking text to semantics

Extracting entities: Linking text to semantics

What are entities?

entity

/ˈɛntɪti/

noun

a thing with distinct and independent existence.

What are **named** entities?

From wiki: Named entity is a real-world object, such as persons, locations, organizations, products, etc., that can be denoted with a proper name. It can be abstract or have a physical existence.

Examples: Donald Trump, Paris, Google.

Detecting entities: Subtasks

Named Entity Recognition: Identify where named entities appear in the text, with particular types (e.g. Organization, Location, Person, Time)

Example: flight to Paris/LOC, set alarm at 8am/Time.

Entity Linking (or disambiguation): Match the identity of the entity mentioned in the text to actual entity (defined by non-ambiguous id)

Example: Donald **Trump** and **Trump** card.

Coreference resolution: Find all expressions that refer to the same entity in a text.

Example: **The food** was salty so guests did not enjoy **it**.

Databases of semantic relations

- ***Is-a***: the relationship between two entities in which one entity inherits from the other.
  a. ***Hypernyms*** are entities with broader/more general meaning,
  b. ***Hyponyms*** are entities with narrower/more specific meaning.

  Example: iPhone is a phone, phone is an electronic device.
- ***Meronymy*** denotes a constituent part of, or a member of something. Meronym is a part of a whole!

  Example: battery is a part of a smartphone.

Databases: WordNet

Databases of semantic relations

- ***Synsets*** denotes synonyms that are interchangeable in some context w/o changing the true value of the statement;

  Example: there were done minor or small mistakes.
- ***Metonymy*** substitution of the name of an attributes for the thing that was meant.

  Example: Business executives are in town VS Suits are in town.
- ***Anything***

  Example: Find who is married to who? What causes what?

Knowledge Databases and Ontologies

Wikipedia: knowledge about the world; not explicitly structured, but often used as a source of entities in the literature.

DBPedia: mined structured content from Wikipedia

WordNet: lexical database for English, representing different word senses as 'synsets' and also containing synset relations.

OMCS: Commonsense knowledge database, representing human knowledge "The sky is blue". Not easy to mine on the web!

YAGO: is an open source knowledge base developed at the Max Planck Institute.

Knowledge Graph

A database of entities and facts (as relations between entities)

Goal: understand/answer **factual** questions about **the world**

Barack Obama

44th U.S. President

barackobama.com

Barack Hussein Obama II is an American politician who served as the 44th President of the United States from January 20, 2009, to January 20, 2017. A member of the Democratic Party, he was the first African American to be elected to the presidency and previously served as a United States Senator from Illinois. Wikipedia

Born: August 4, 1961 (age 57 years), Kapiʻolani Medical Center for Women and Children, Honolulu, Hawaii, United States

Height: 1.85 m

Vice president: Joe Biden (2009–2017)

Education: Harvard Law School (1988–1991), MORE

Parents: Ann Dunham, Barack Obama Sr.

Entity Linking: Common approach

1. Annotate text
2. Find entity candidates by their surface forms (= n-grams of how an entity can appear in the text)
3. Disambiguate to particular entity based on the context (e.g. co-occurring words)

Advanced NLP 2: Sequence Modeling

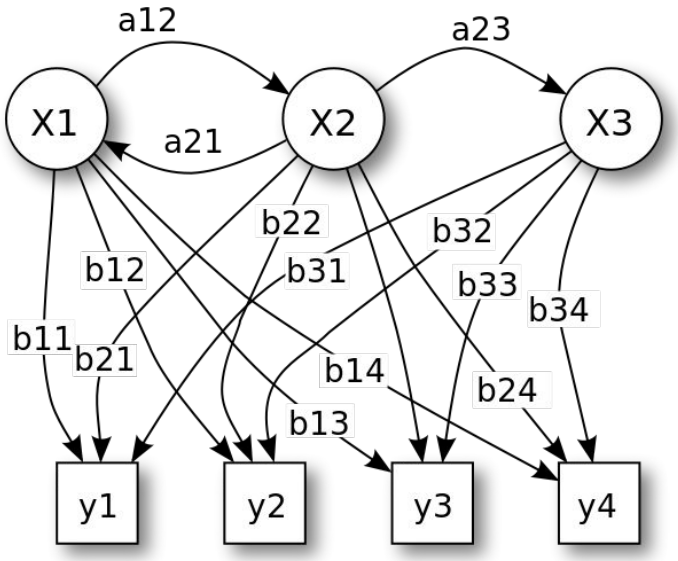
Sequence Modeling (HMM, CRF)

Hidden Markov Models

$$p(l, s) = p(l_1) \prod_i p(l_i|l_{i-1}) p(w_i|l_i)$$

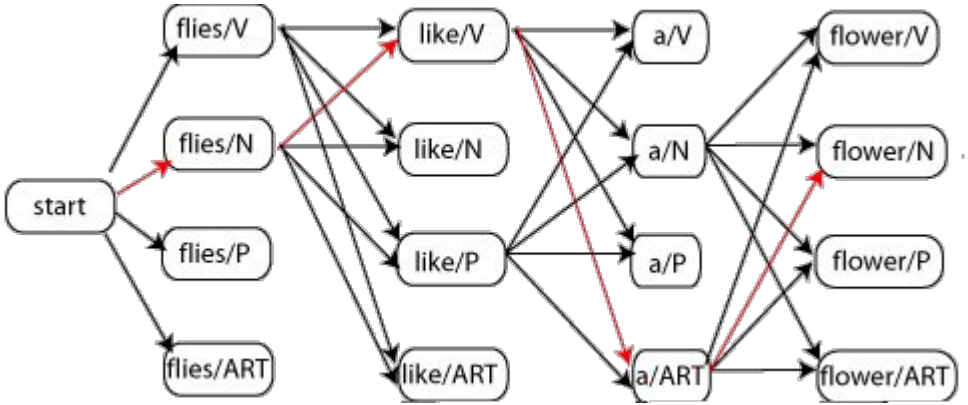
where

- $p(l_i|l_{i-1})$ are transition probabilities
- $p(w_i|l_i)$ are emission probabilities (e



Hidden Markov Models

POS example:



Conditional Random Fields

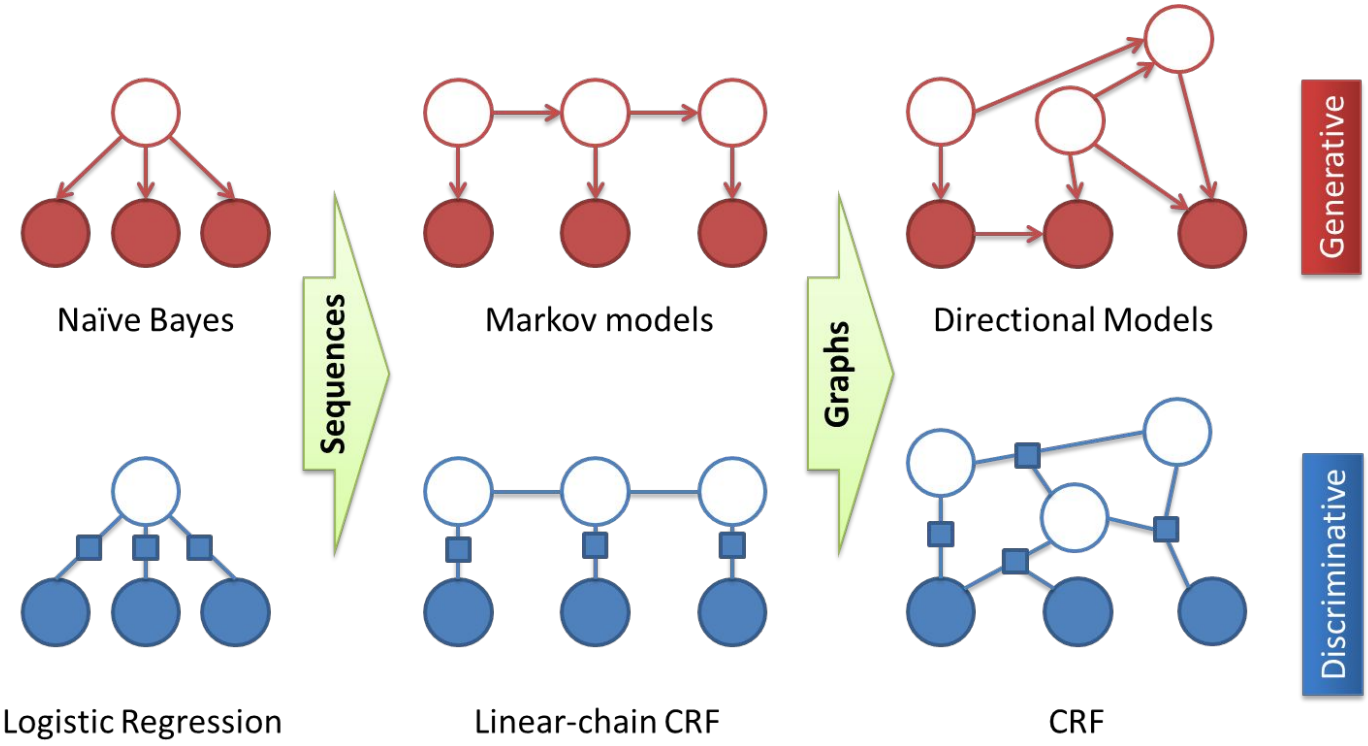
Feature function is a function that takes in as input:

- a sentence s
- the position i of a word in the sentence
- the label Li of the current word
- the label Li-1 of the previous word

So given a sequence s we can now score it as follows $$score(l|s) = \sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_j f_j(s, i, l_i, l_{i-1})$$

$$p(l|s) = \frac{exp[score(l|s)]}{\sum_{l'} exp[score(l'|s)]} = \frac{exp[\sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_j f_j(s,i,l_i,l_{i-1})]}{\sum_{l'} exp[\sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_j f_j(s,i,l'_i,l'_{i-1})]}$$

Arbitrary features can be also added on top :)



Adapted from C. Sutton, A. McCallum, "An Introduction to Conditional Random Fields", ArXiv, November 2010

Sequence Modeling

Motivation: Why Sequence Models?

Recurrent Neural Networks: Overview

Recurrent Neural Network Architectures

Types of RNN Cells

Outline

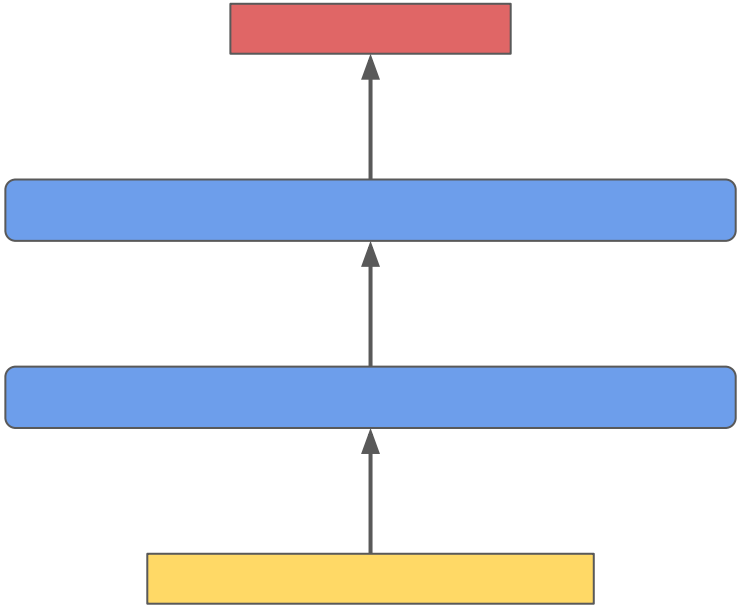
Motivation: Why Sequence Models?

Recurrent Neural Networks: Overview

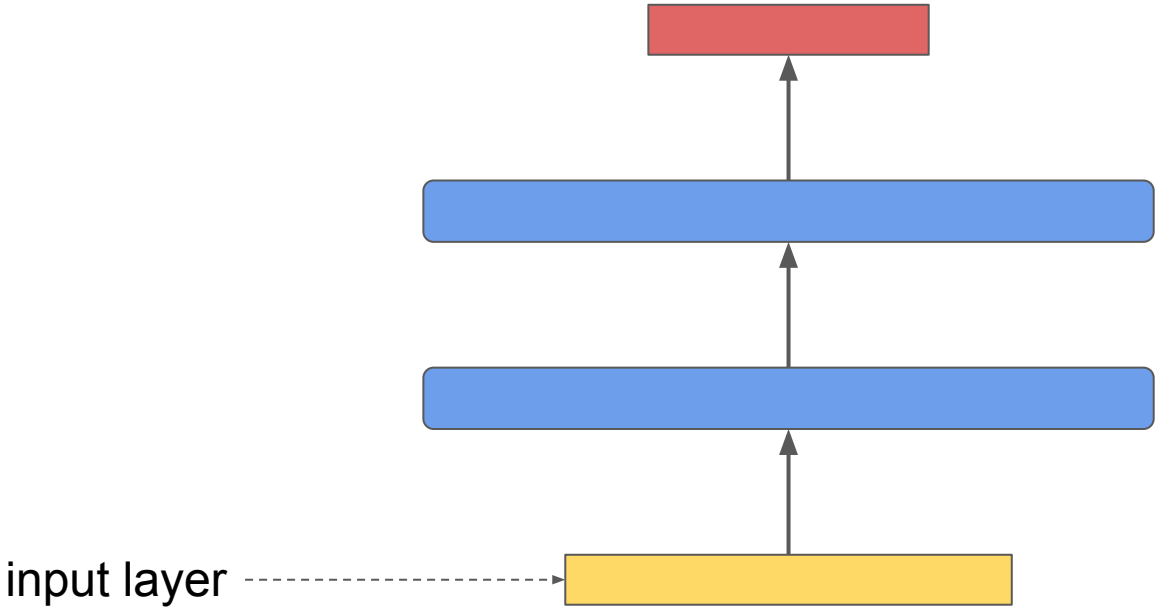
Recurrent Neural Network Architectures

Types of RNN Cells

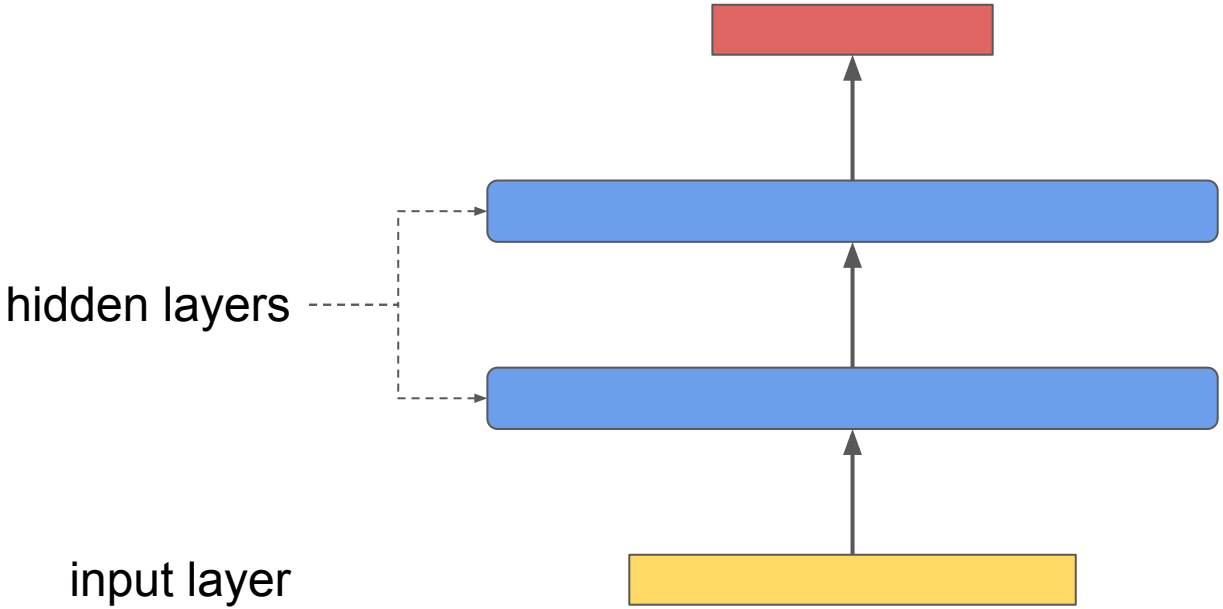
Feed Forward Networks



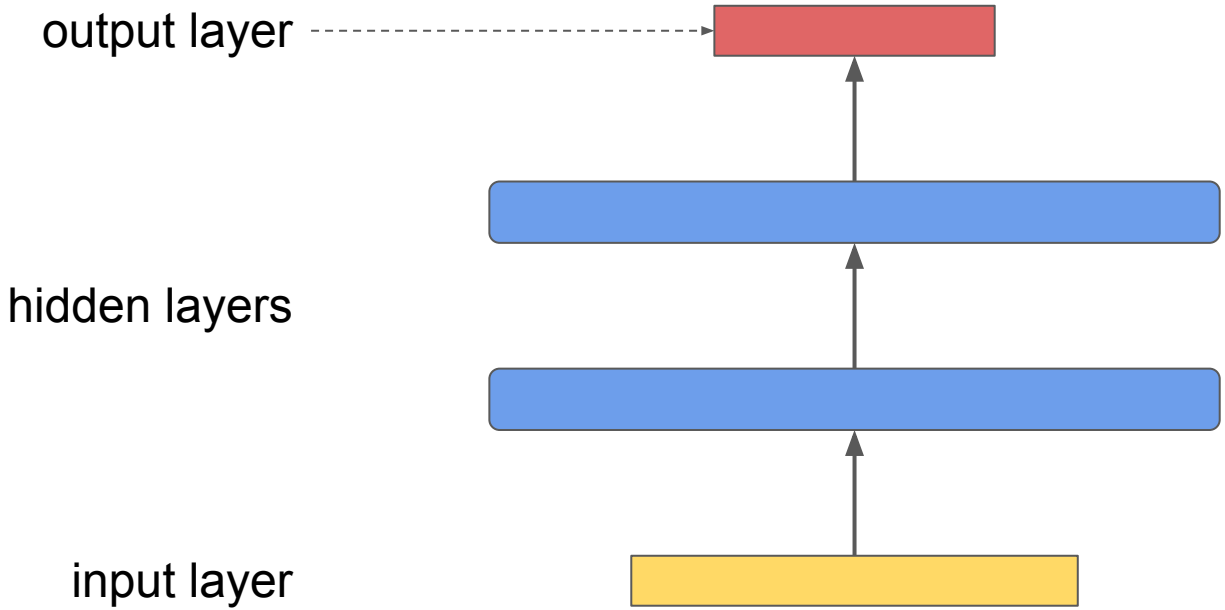
Feed Forward Networks



Feed Forward Networks

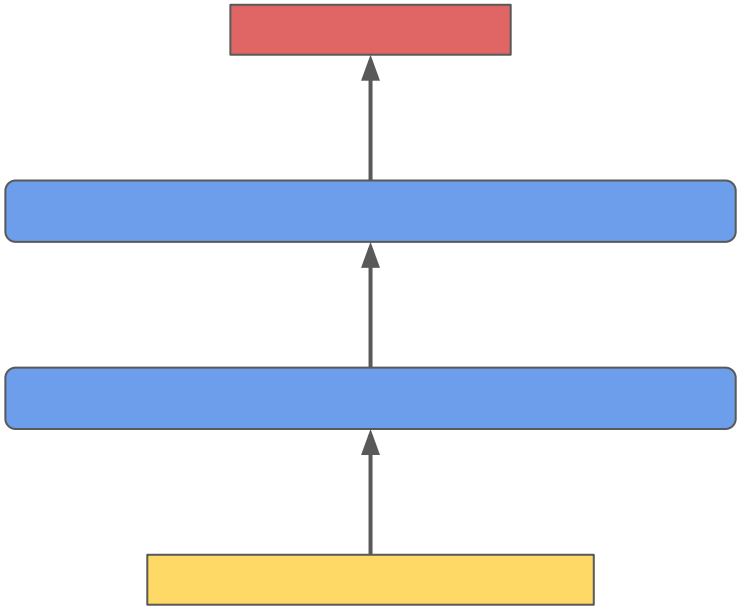


Feed Forward Networks



Feed Forward Networks

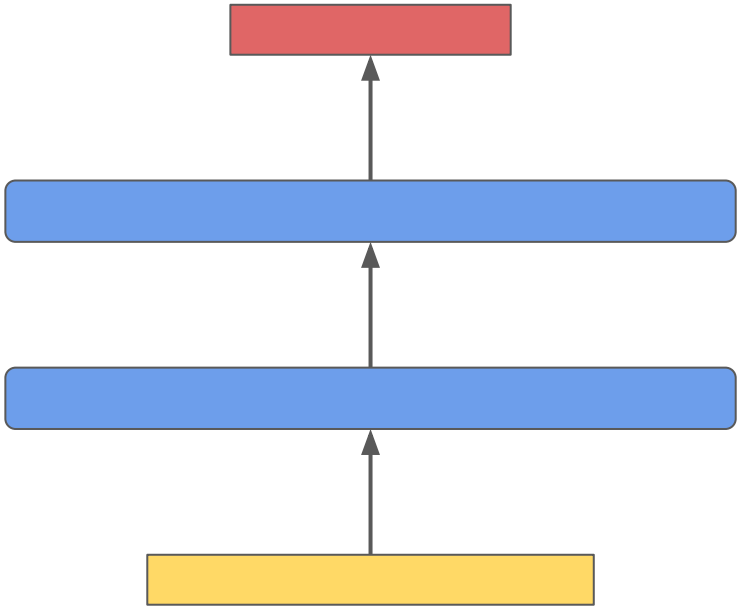
Fixed size layers



Feed Forward Networks

Fixed size layers

Inference is stateless

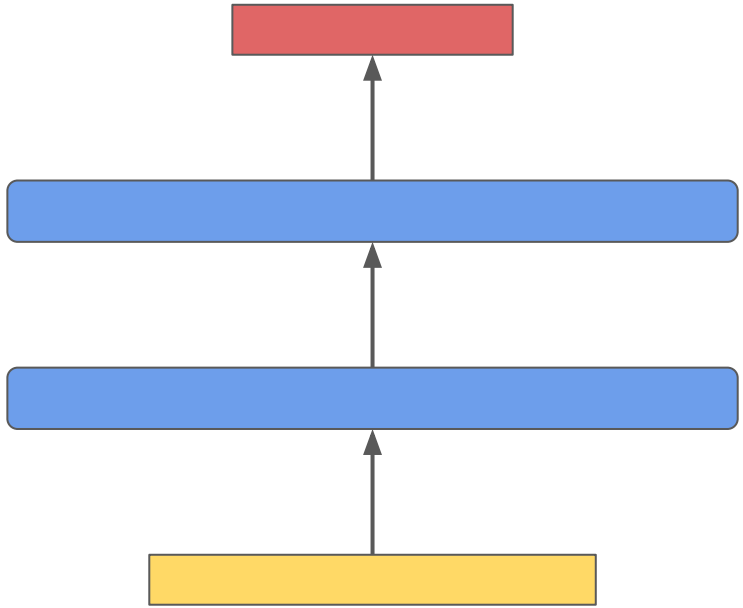


Feed Forward Networks

Fixed size layers

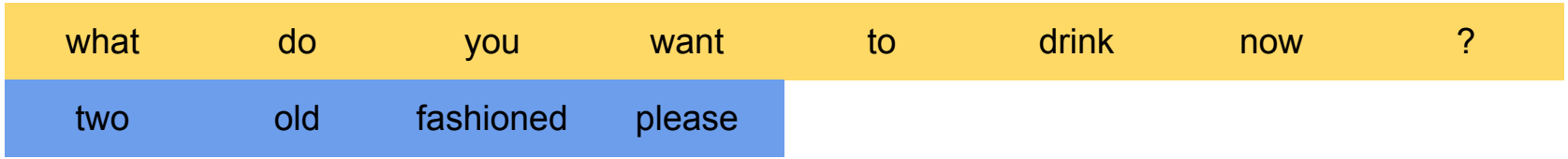
Inference is stateless

Nodes are unordered

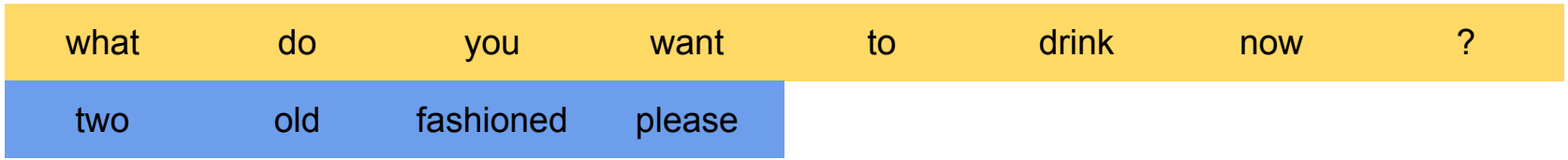


Language as Input

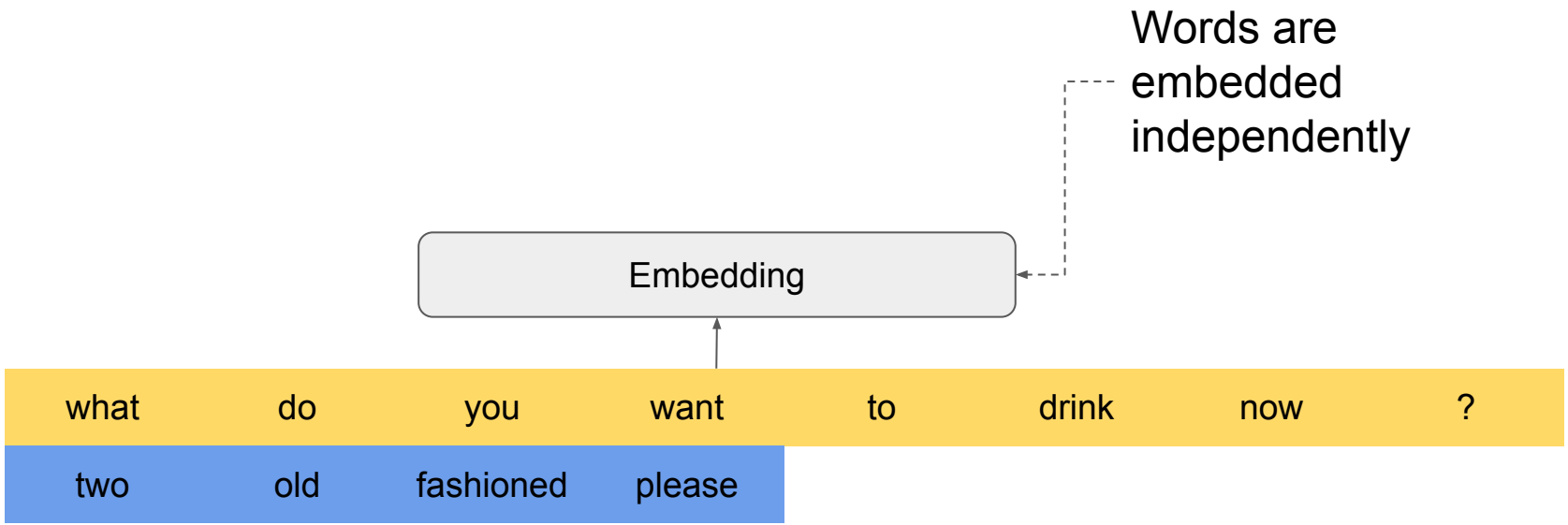
Input to a language model can have variable length. For example,



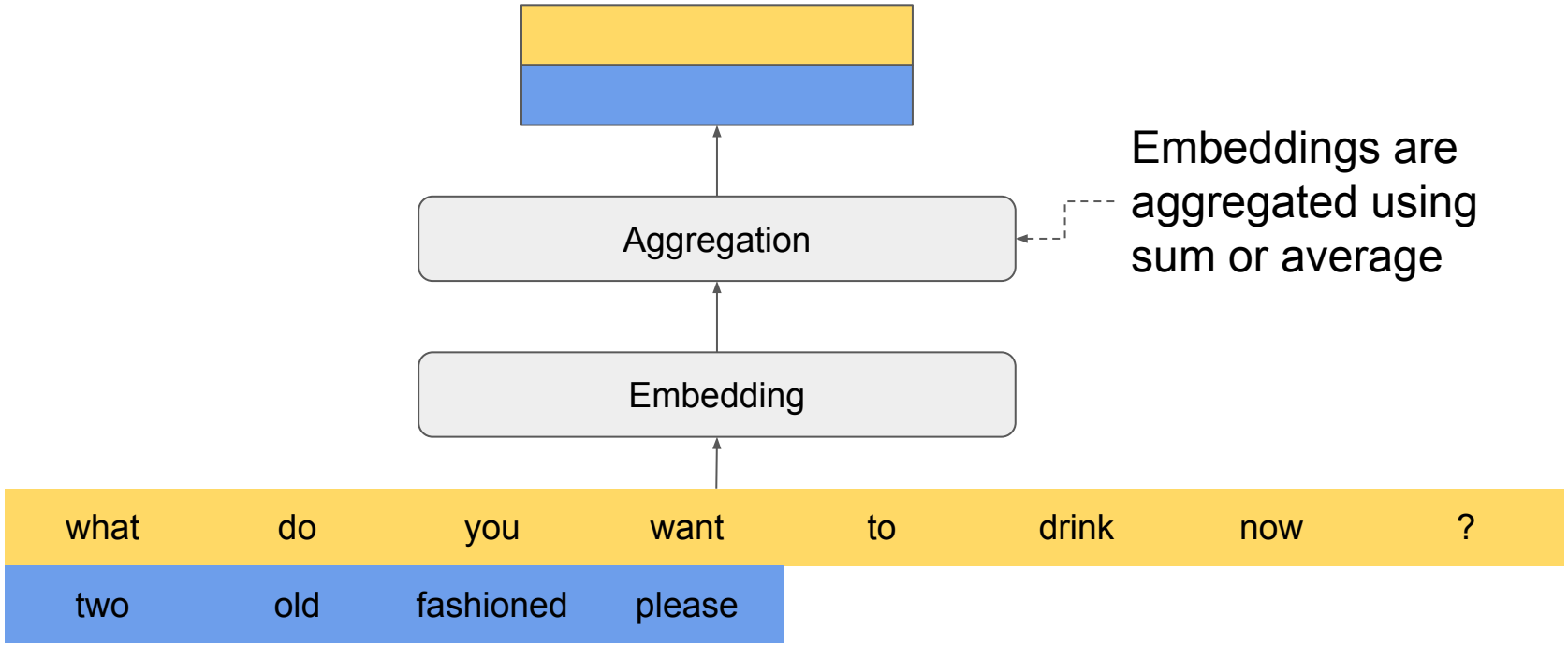
Language as Input: the “Typical” Approach



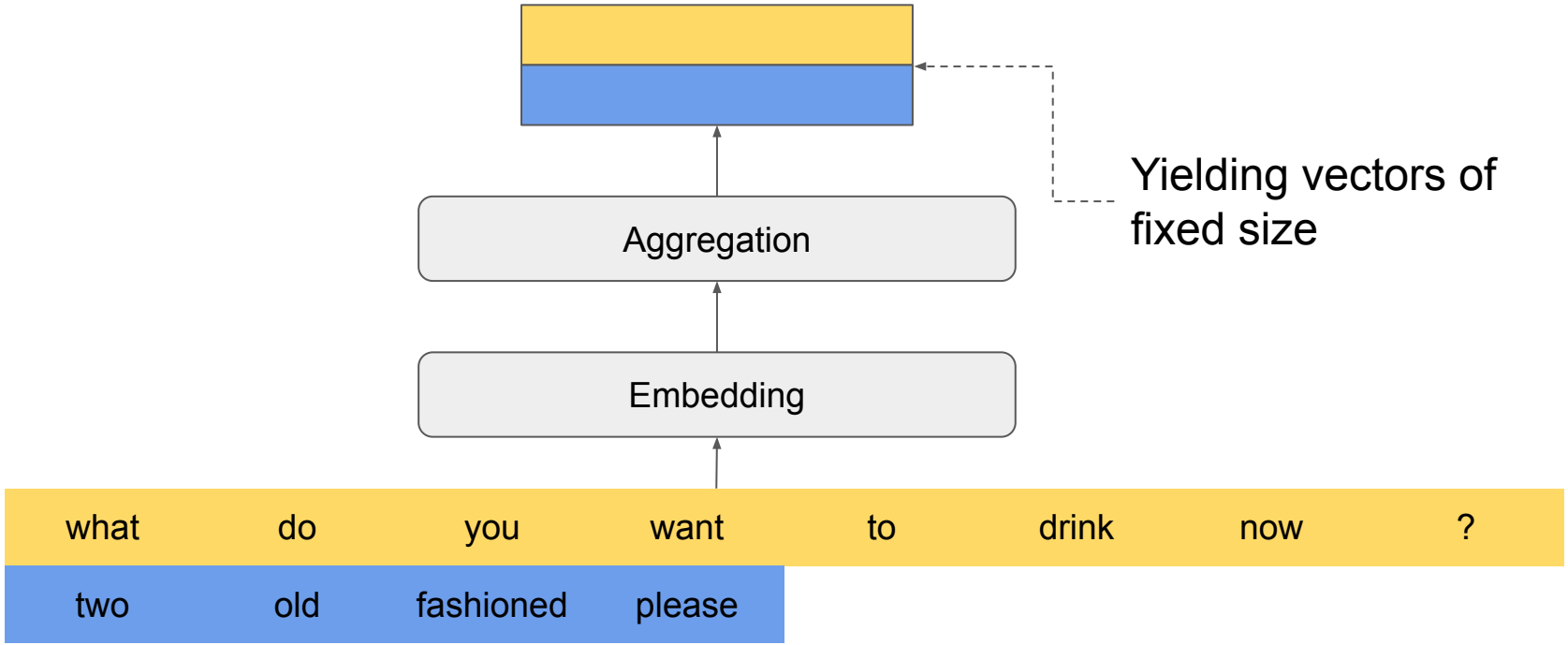
Language as Input: the “Typical” Approach



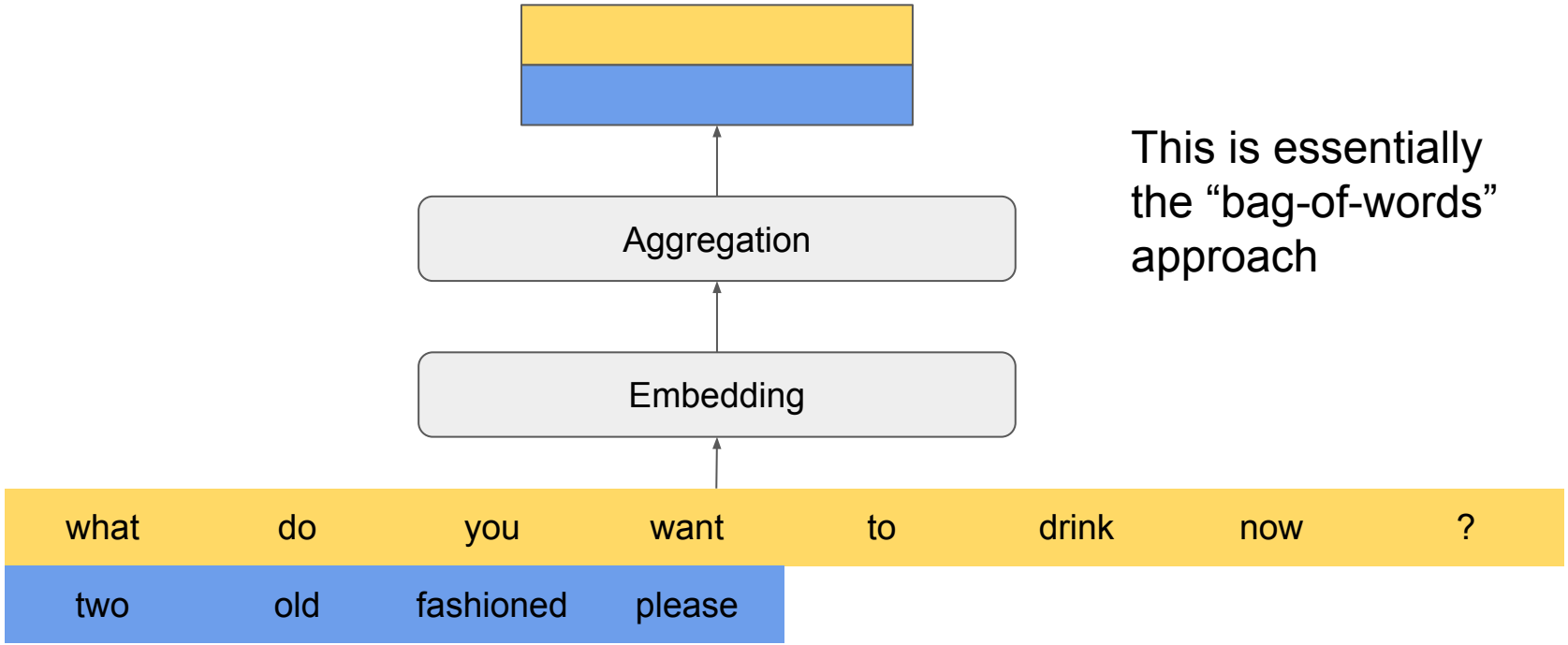
Language as Input: the “Typical” Approach



Language as Input: the “Typical” Approach



Language as Input: the “Typical” Approach



This is essentially the “bag-of-words” approach

Bag of Words: Pros and Cons

Pro: Allows variable length data to be fed into a typical feed-forward network.

Pro: Variable length features can easily be combined with fixed length features.

Pro: Widespread empirical success.

Bag of Words: Pros and Cons

Pro: Allows variable length data to be fed into a typical feed-forward network.

Pro: Variable length features can easily be combined with fixed length features.

Pro: Widespread empirical success.

Con: Destroys the structure of the input:

The cat sat on the mat != {cat, sat, the, the, mat, on}

Structure and Ordering is Important

For example:

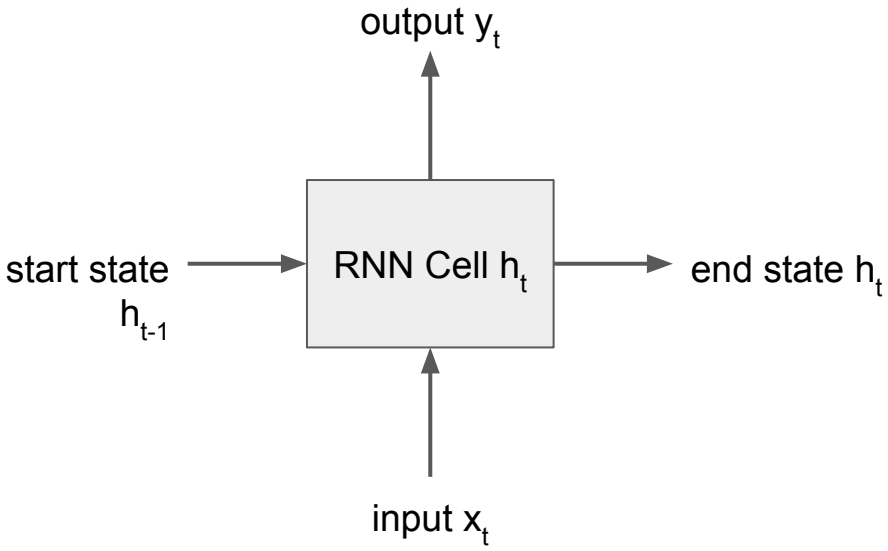
Humor or sarcasm

Modifiers such as somewhat, extremely or not

Multi-word phrases such as tiger mosquito or zurich city bridge

Big Idea of Recurrent Neural Networks

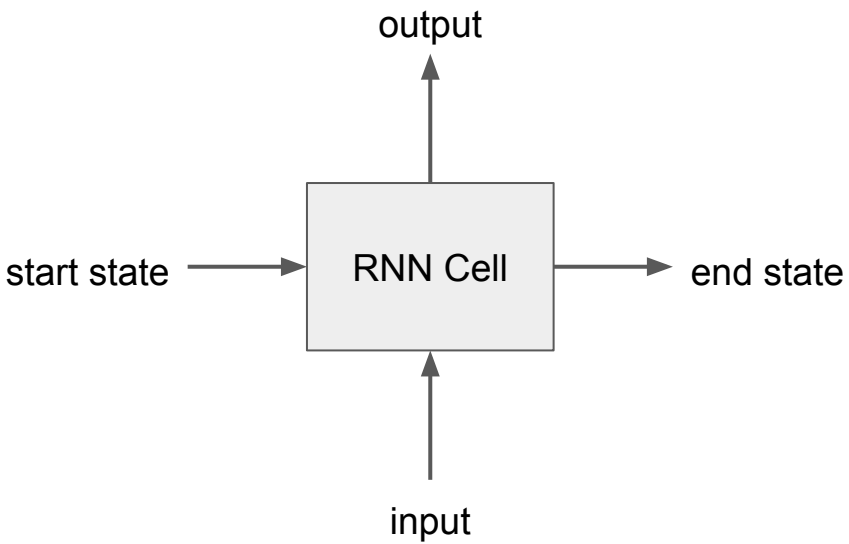
Recurrent Neural Nets: A Cell...




```
output, end_state =
  rnn_cell(input, start_state)
```

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

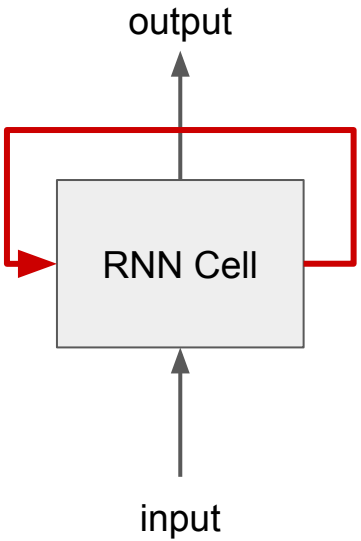
Recurrent Neural Nets: A Cell...



The only rule:

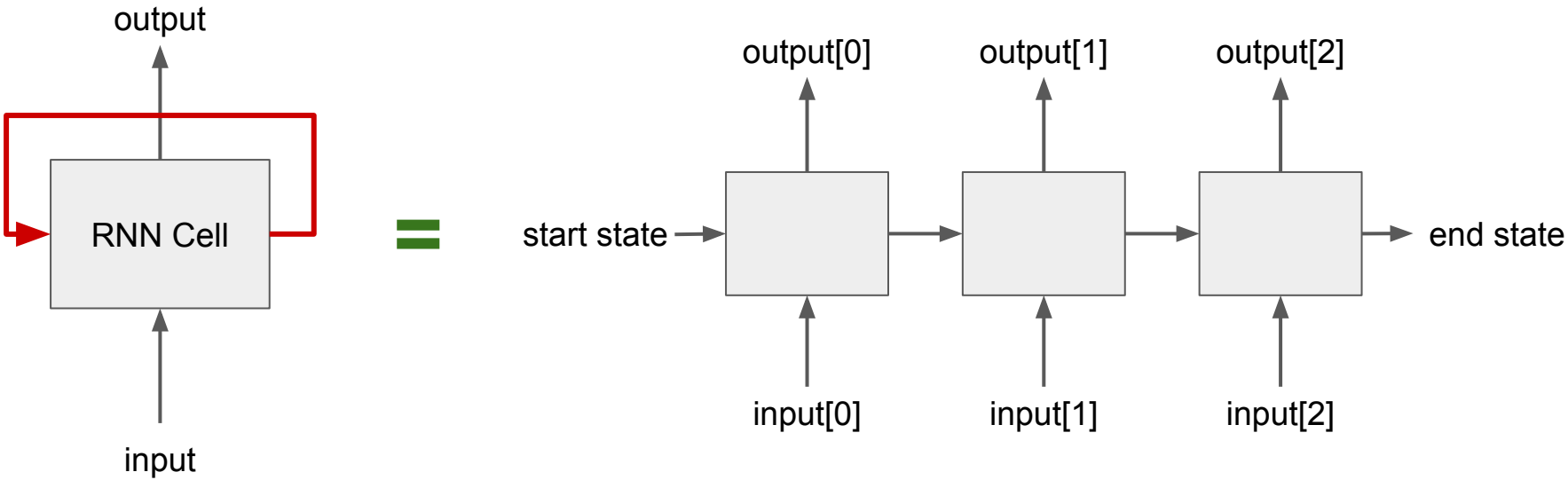
start state and end state must have the same dimension.

Recurrent Neural Nets: A Cell...Wrapped in a For Loop



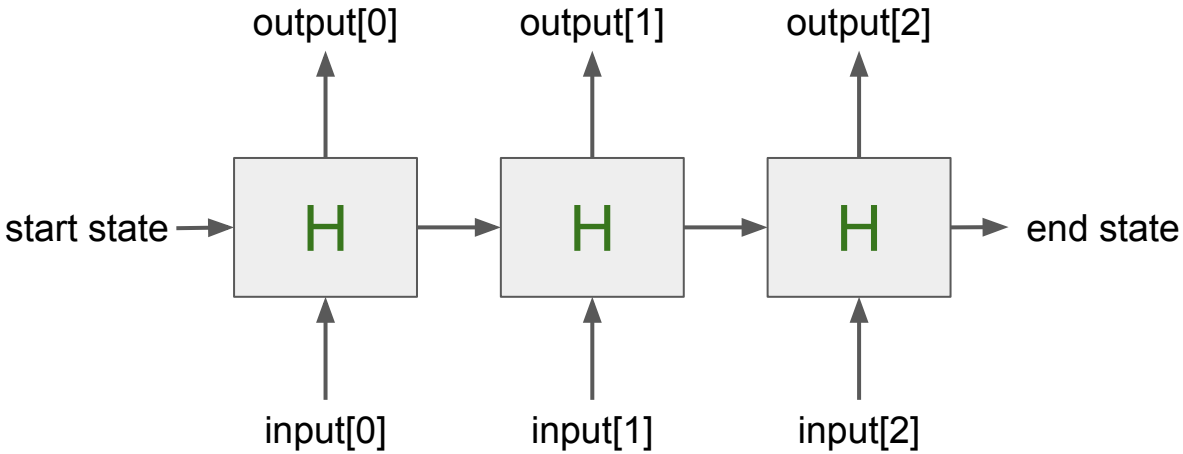

```
for i in range(sequence_length):

  output[i], end_state =
    rnn_cell(input[i], start_state)

  start_state = end_state
```

Recurrent Neural Nets: Unrolling the For Loop



Recurrent Neural Nets: Unrolling the For Loop

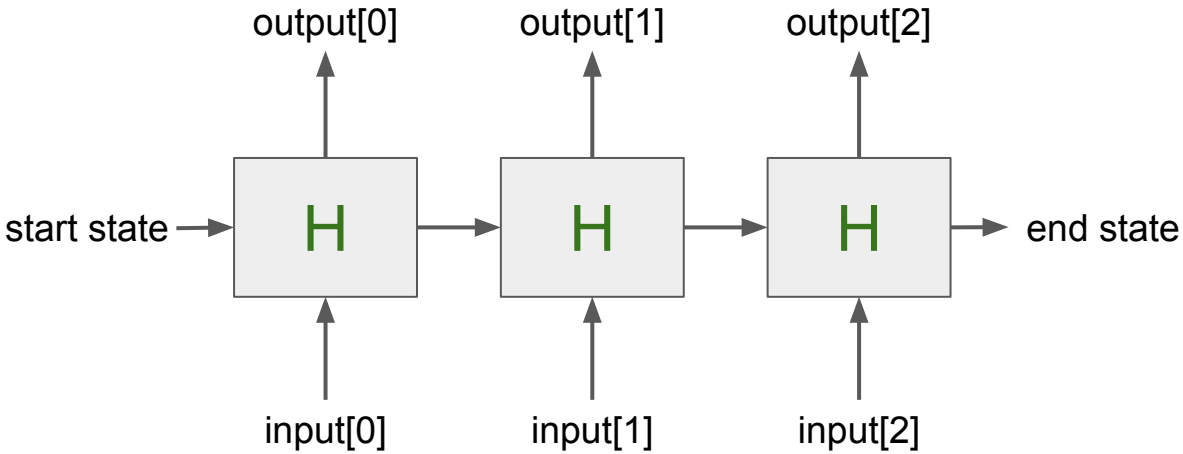
Each cell is identical and they share weights.



Recurrent Neural Nets: Unrolling the For Loop

Each cell is identical and they share weights.

When we train an RNN, we are really training a single cell that is used repeatedly.

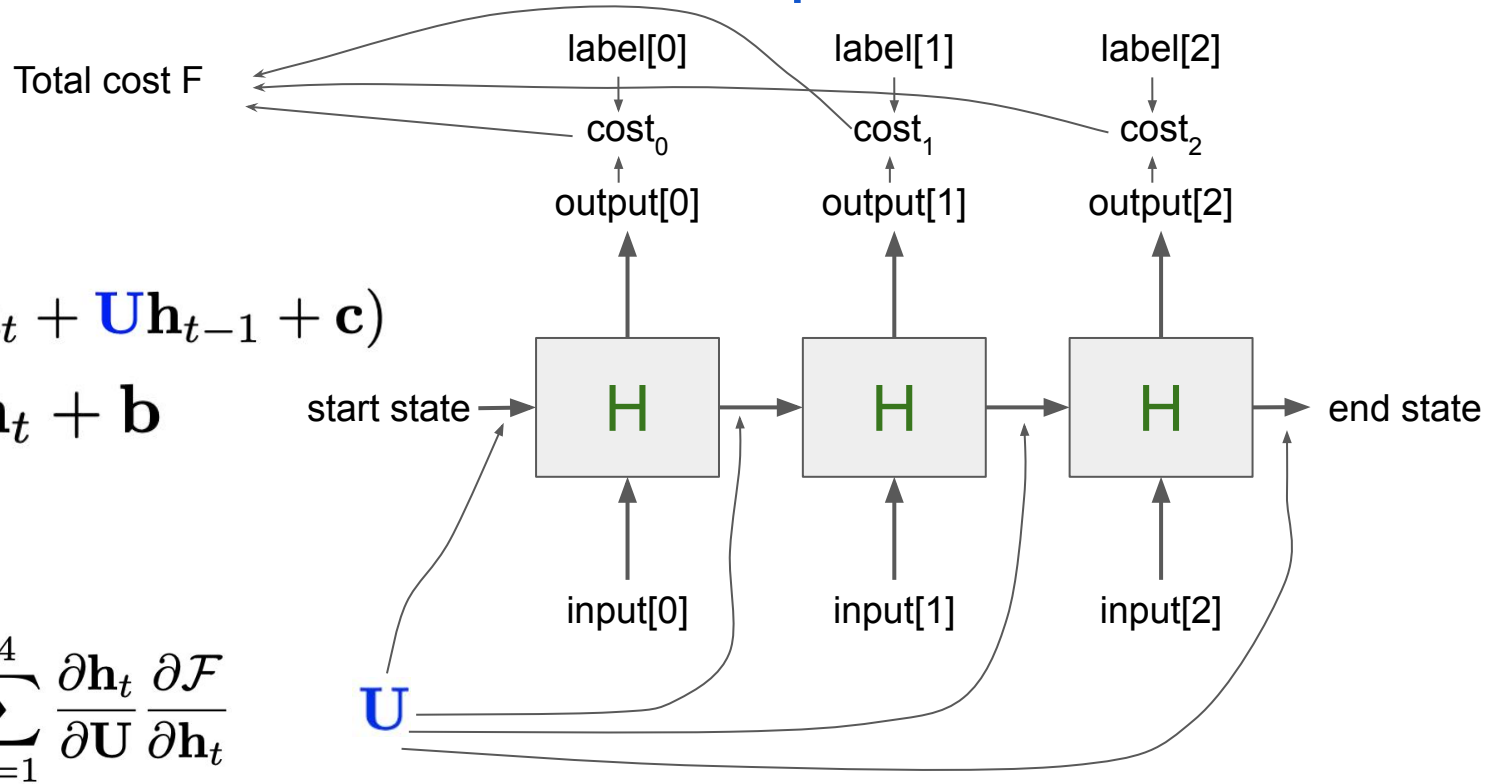


Recurrent Neural Nets: Error computation

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

$$\frac{\partial \mathcal{F}}{\partial \mathbf{U}} = \sum_{t=1}^{4} \frac{\partial \mathbf{h}_t}{\partial \mathbf{U}} \frac{\partial \mathcal{F}}{\partial \mathbf{h}_t}$$



Outline

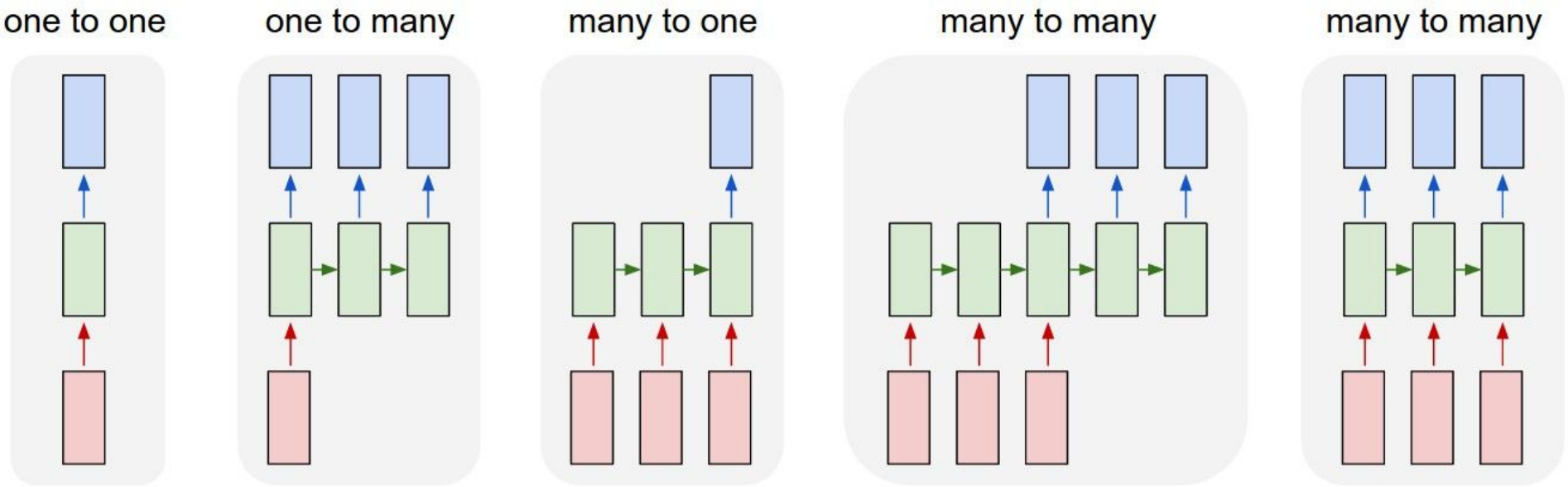
Motivation: Why Sequence Models?

Recurrent Neural Networks: Overview

Recurrent Neural Network Architectures

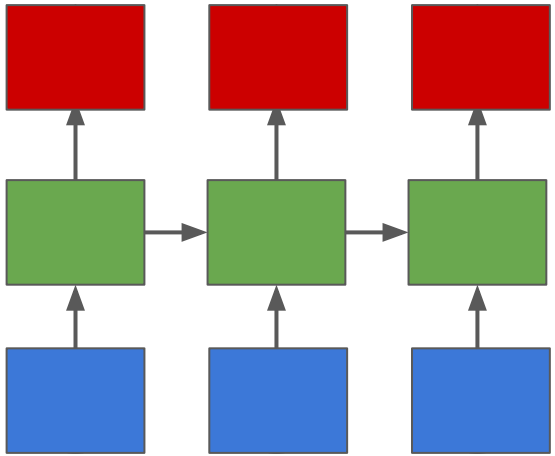
Types of RNN Cells

Types Of NNs



Taken from [2].

RNN Architectures: Many to Many

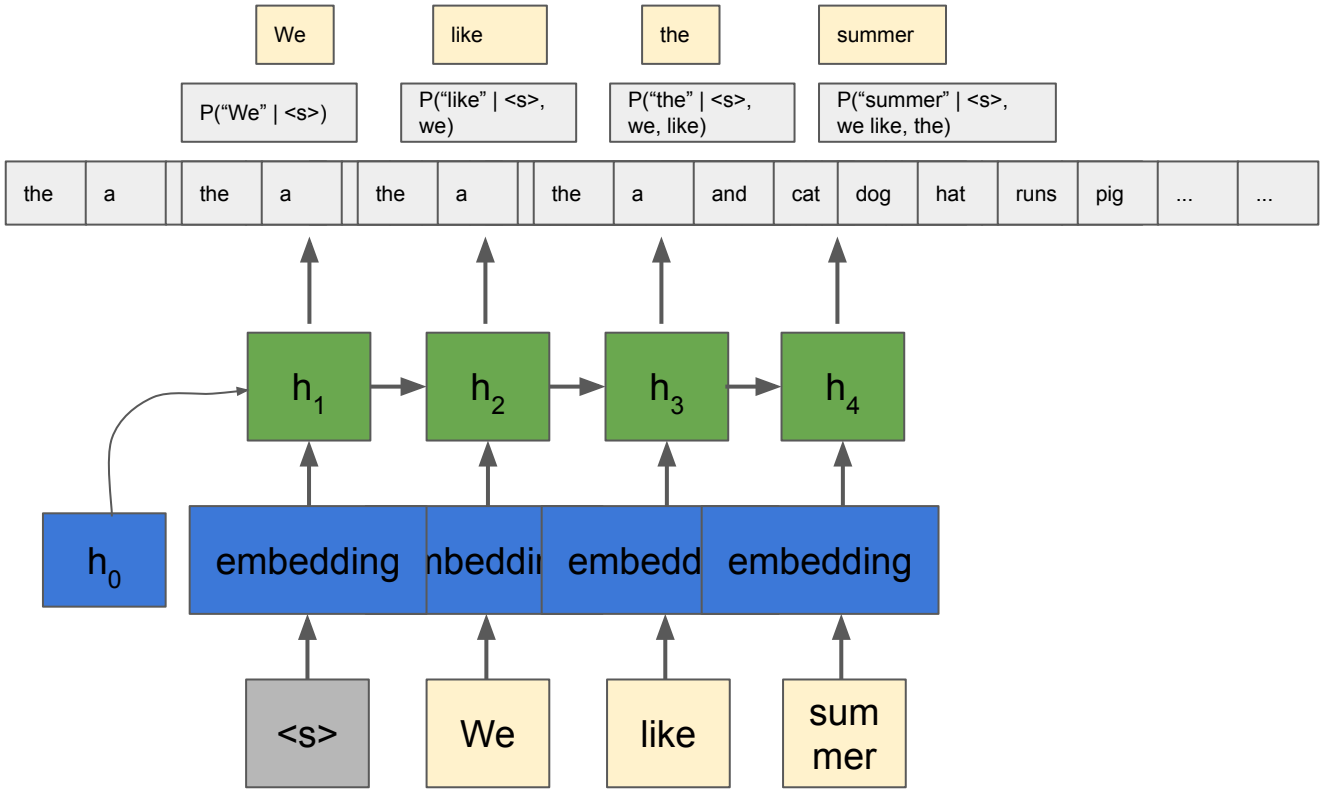


Example: speech recognition

Input: windowed audio

Output: sequence of letters

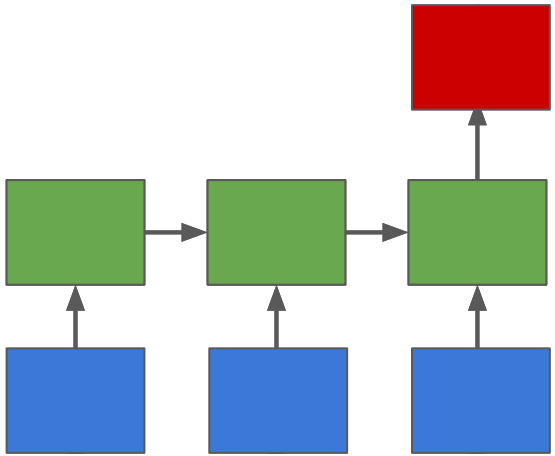
RNN Architectures: Example



$$\mathbf{u} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

$$p_i = \frac{\exp u_i}{\sum_j \exp u_j}$$

RNN Architectures: Many to One

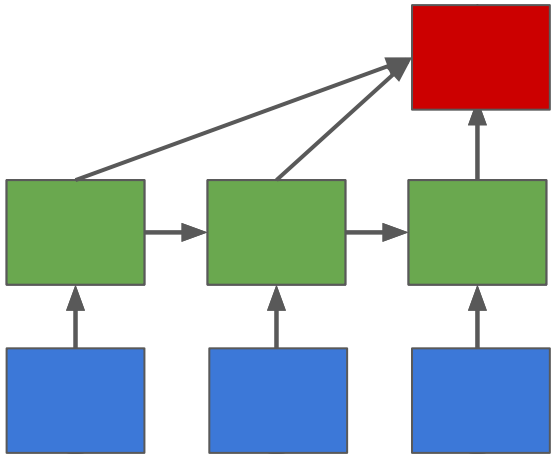


Example: spam filtering

Input: words in an email

Output: spam or not spam

RNN Architectures: Many to One

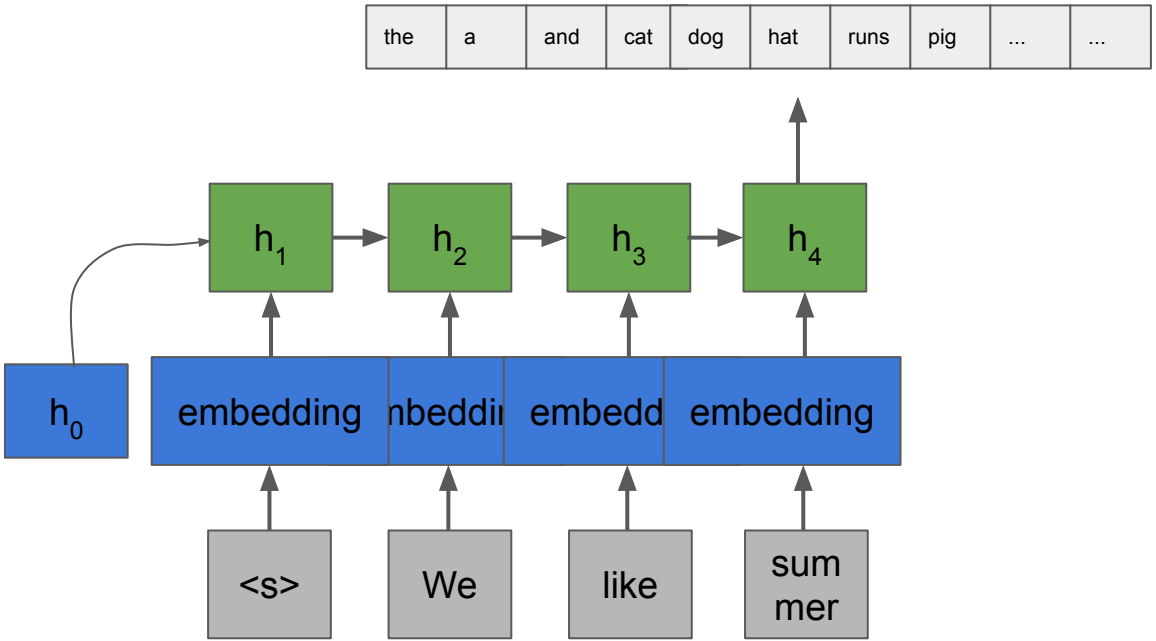


Example: spam filtering

Input: words in an email

Output: spam or not spam

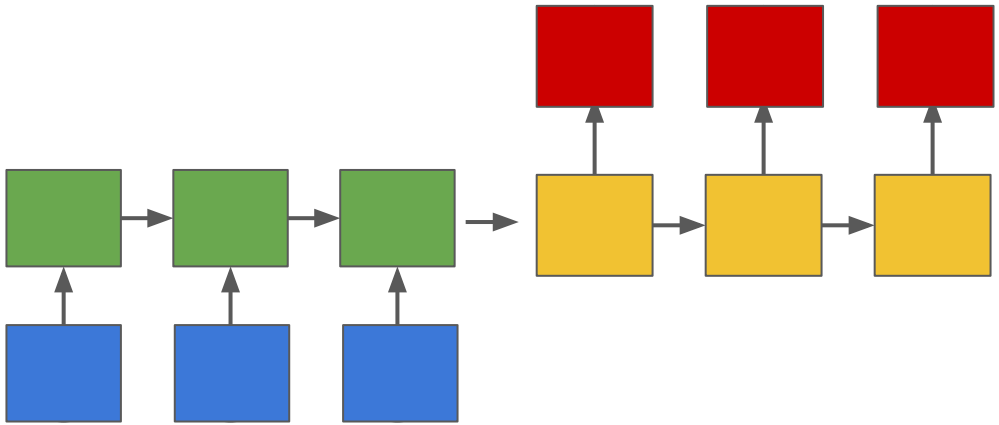
RNN Architectures: Example



$$\mathbf{u} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

$$p_i = \frac{\exp u_i}{\sum_j \exp u_j}$$

RNN Architectures: Sequence to Sequence

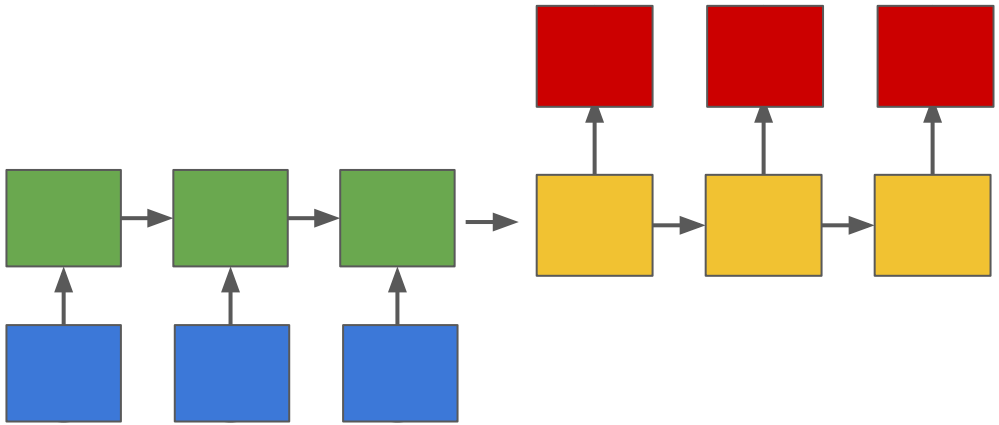


Example: translation

Input: Spanish

Output: Mandarin

RNN Architectures: Sequence to Sequence

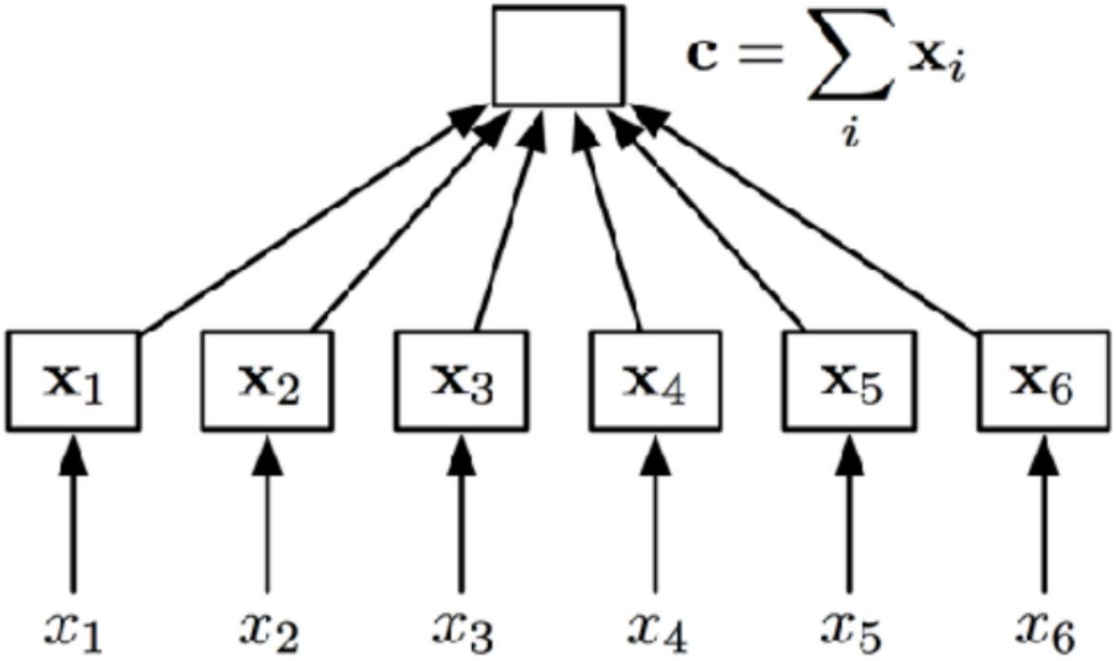


We often use different cells for encoding and decoding.

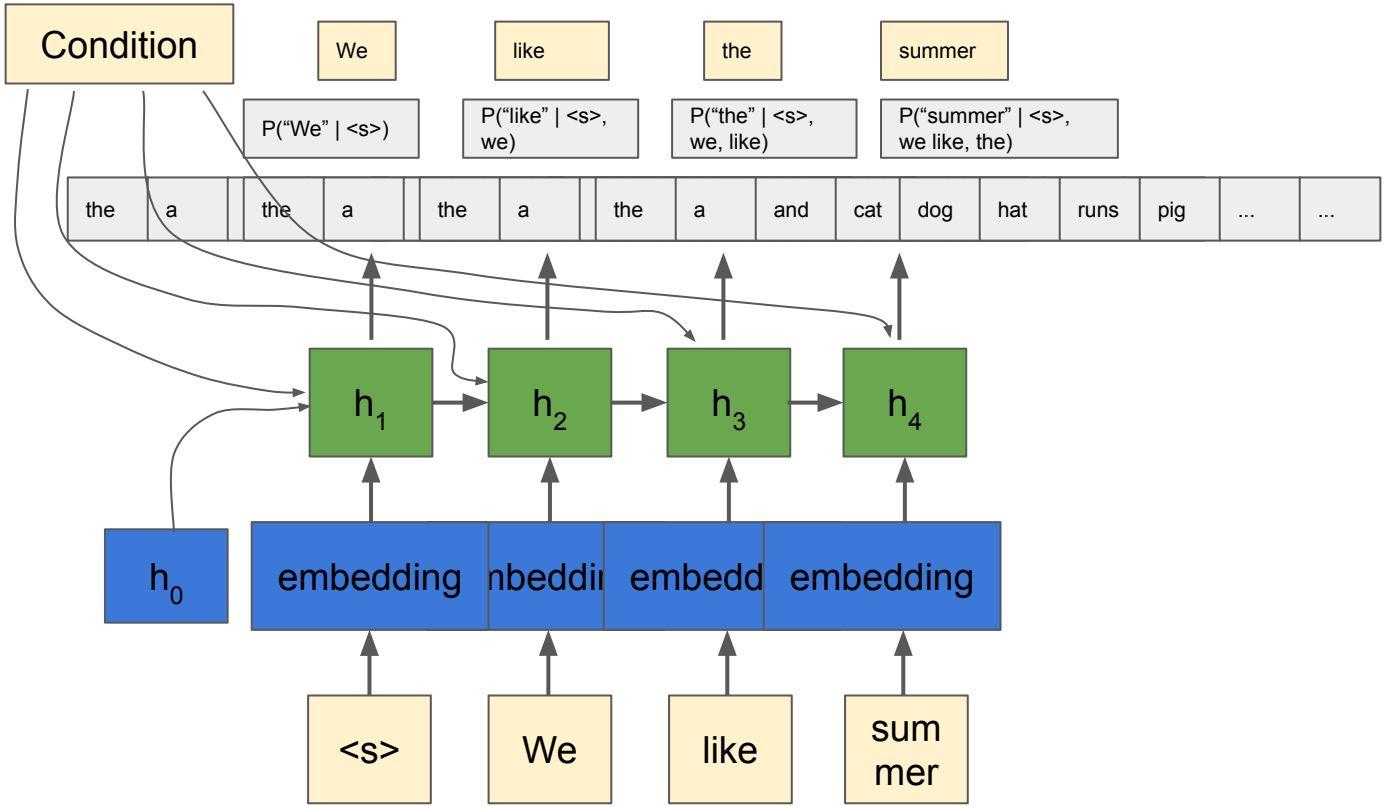
Conditional LM

x "input"	w "**text** output"
An author	A document written by that author
A topic label	An article about that topic
{SPAM, NOT_SPAM}	An email
A sentence in French	Its English translation
A sentence in English	Its French translation
A sentence in English	Its Chinese translation
An image	A text description of the image
A document	Its summary
A document	Its translation
Meterological measurements	A weather report
Acoustic signal	Transcription of speech
Conversational history + database	Dialogue system response
A question + a document	Its answer
A question + an image	Its answer

Encoder


$\mathbf{c} = \sum_i \mathbf{x}_i$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_5$
$\mathbf{x}_6$
x_1
x_2
x_3
x_4
x_5
x_6

RNN Architectures: Example



RNN Architectures: Beam Search

E.g., for b=2:

$$\boldsymbol{x} = Bier\ trinke\ ich$$

beer **drink** **I**

$\langle s \rangle$
logprob=0

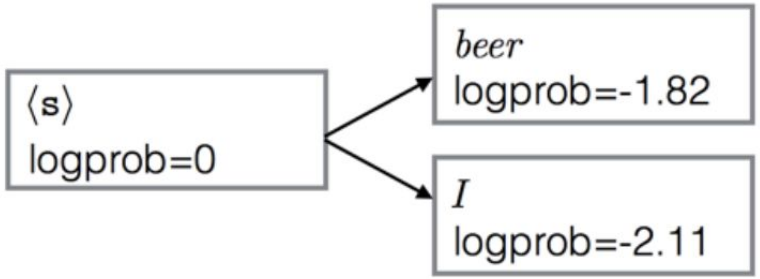
w_0 w_1 w_2 w_3

RNN Architectures: Beam Search

E.g., for b=2:

$$\boldsymbol{x} = Bier\ trinke\ ich$$

beer **drink** **I**



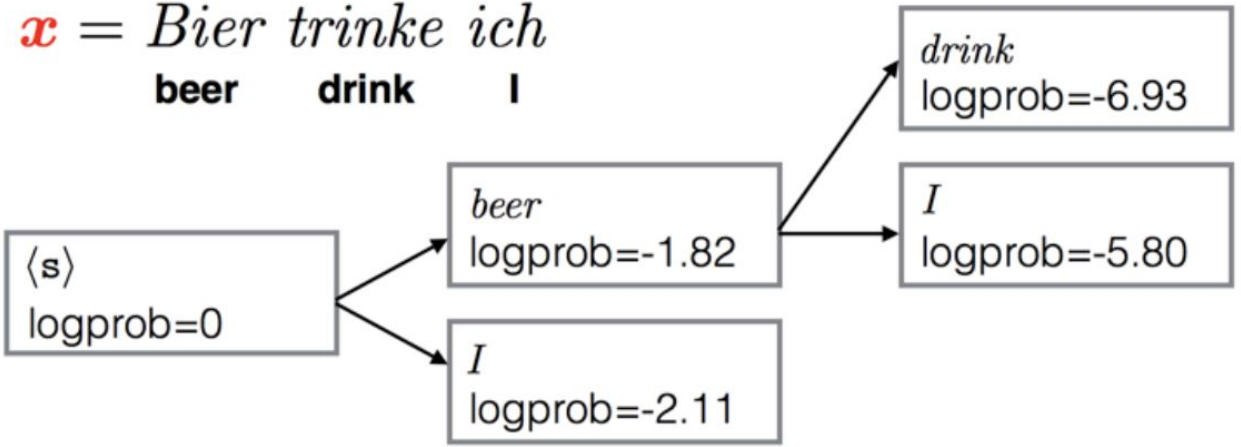
w_0 w_1 w_2 w_3

RNN Architectures: Beam Search

E.g., for *b*=2:

$\boldsymbol{x} = Bier\ trinke\ ich$

beer **drink** **I**



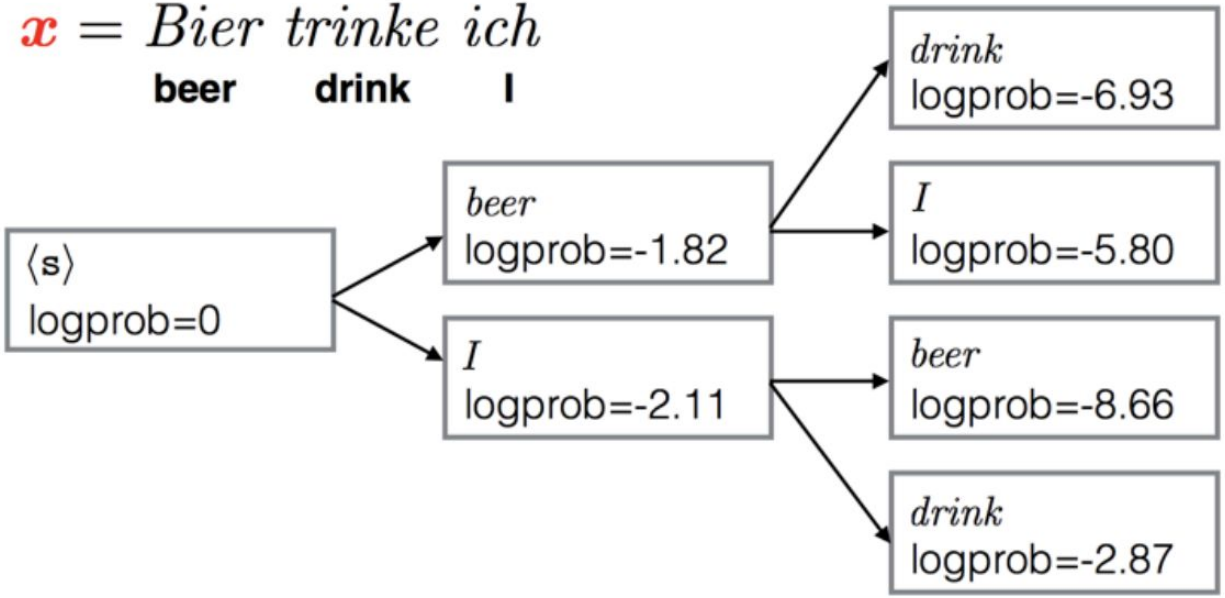
w_0 w_1 w_2 w_3

RNN Architectures: Beam Search

E.g., for b=2:

$\boldsymbol{x} = Bier\ trinke\ ich$

beer **drink** **I**

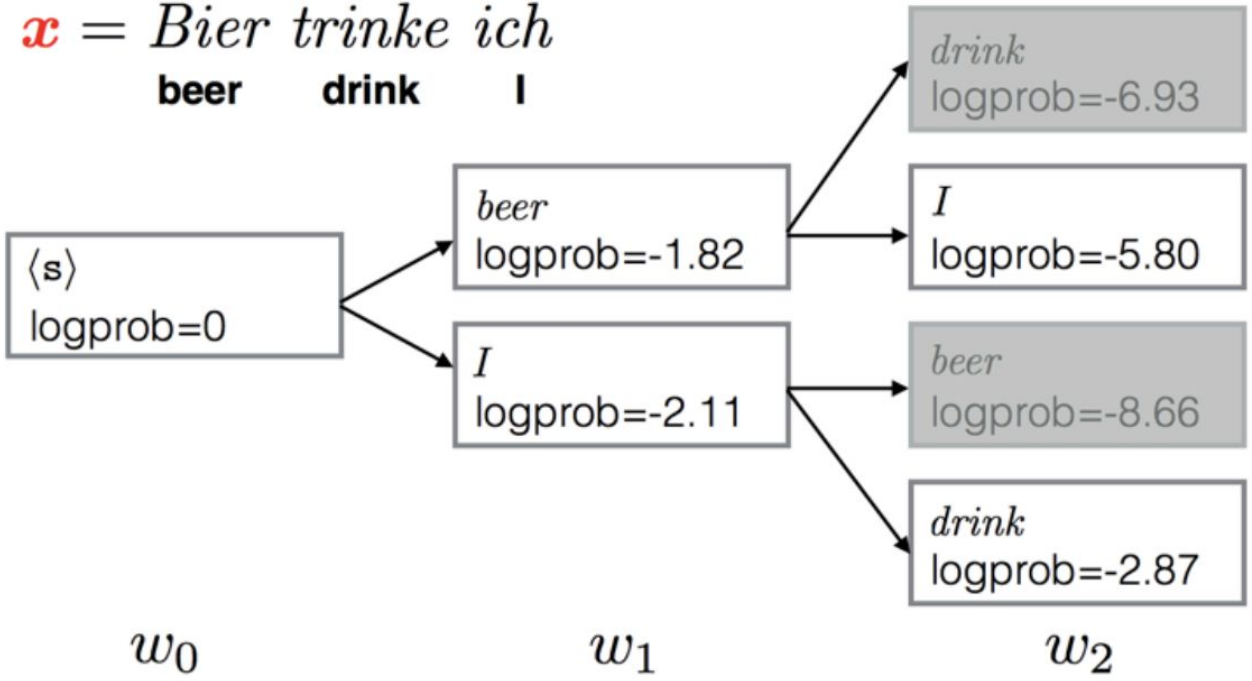


RNN Architectures: Beam Search

E.g., for b=2:

$\boldsymbol{x} = Bier\ trinke\ ich$

beer **drink** **I**

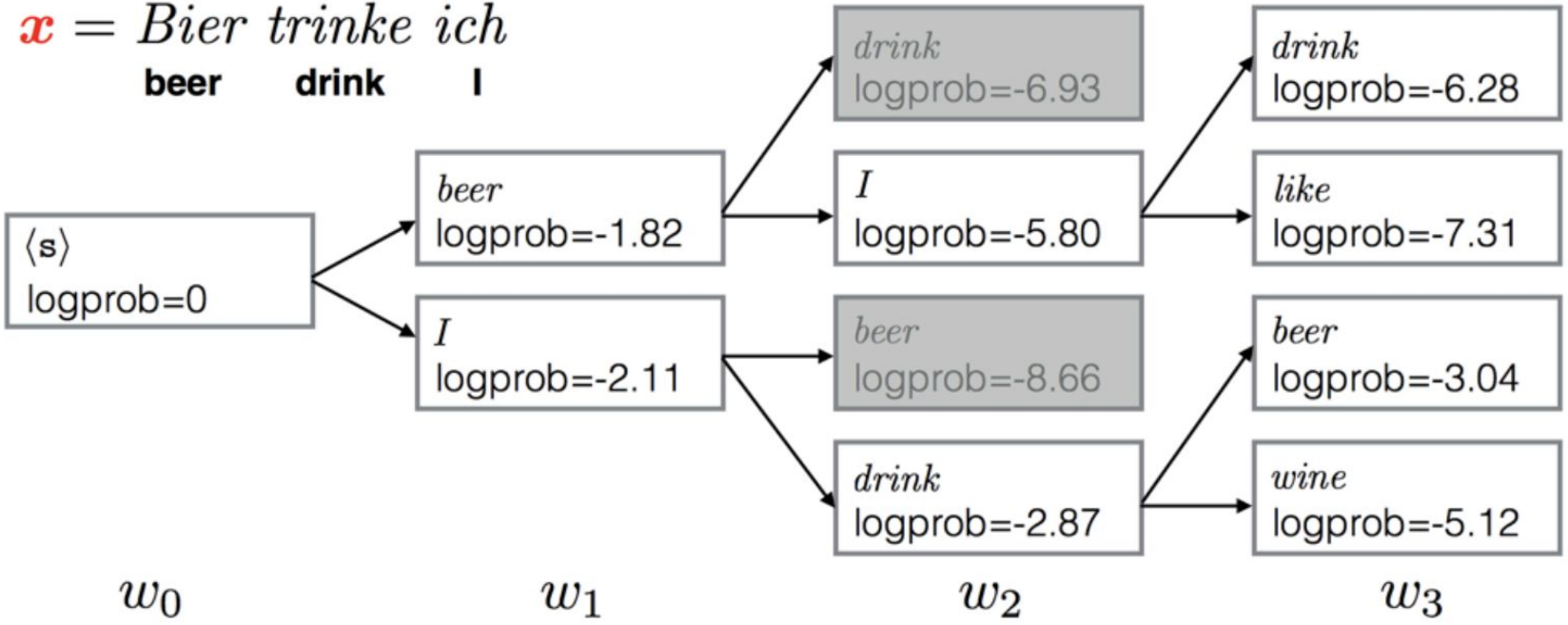


RNN Architectures: Beam Search

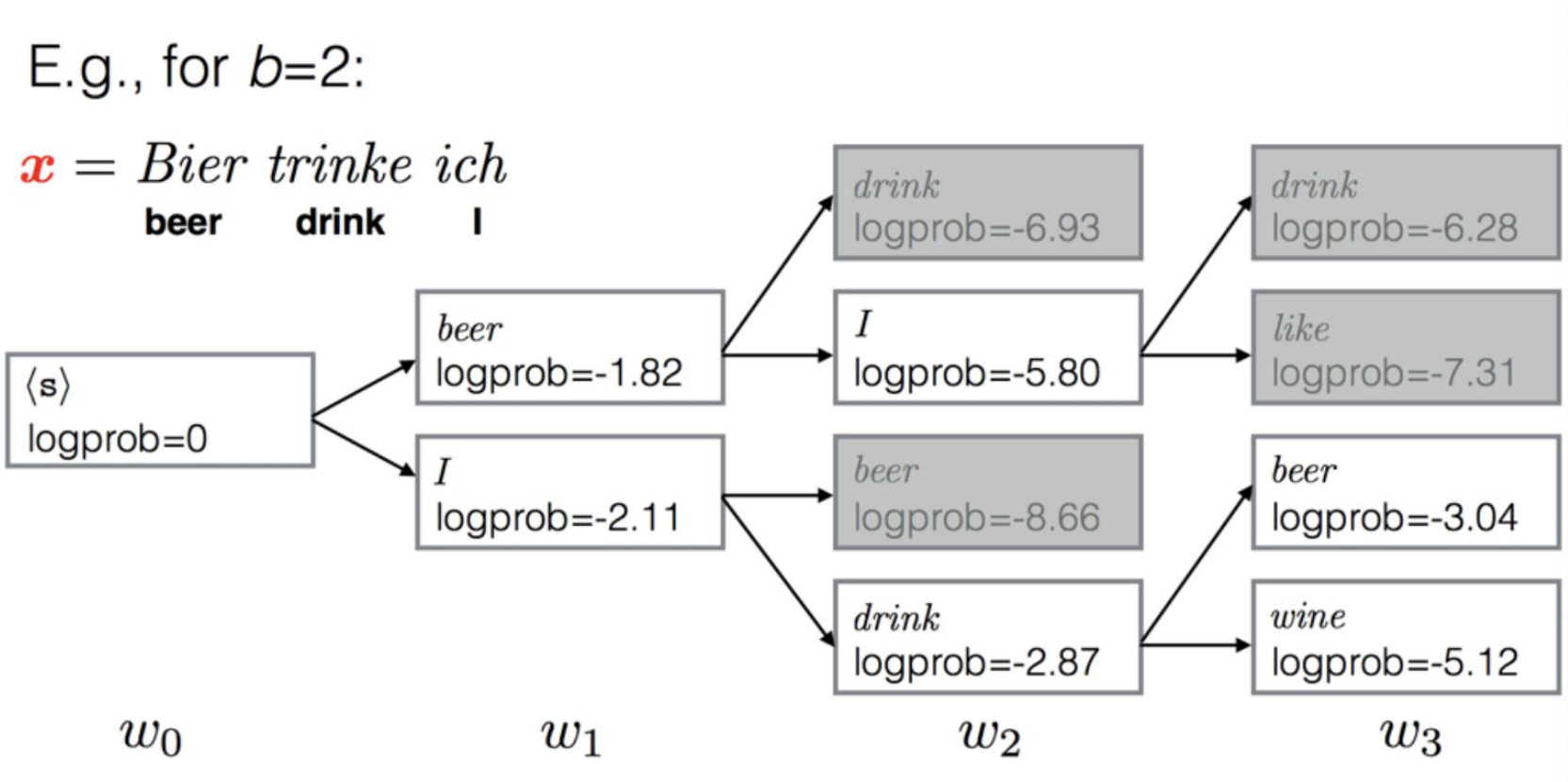
E.g., for b=2:

$\boldsymbol{x} = Bier\ trinke\ ich$

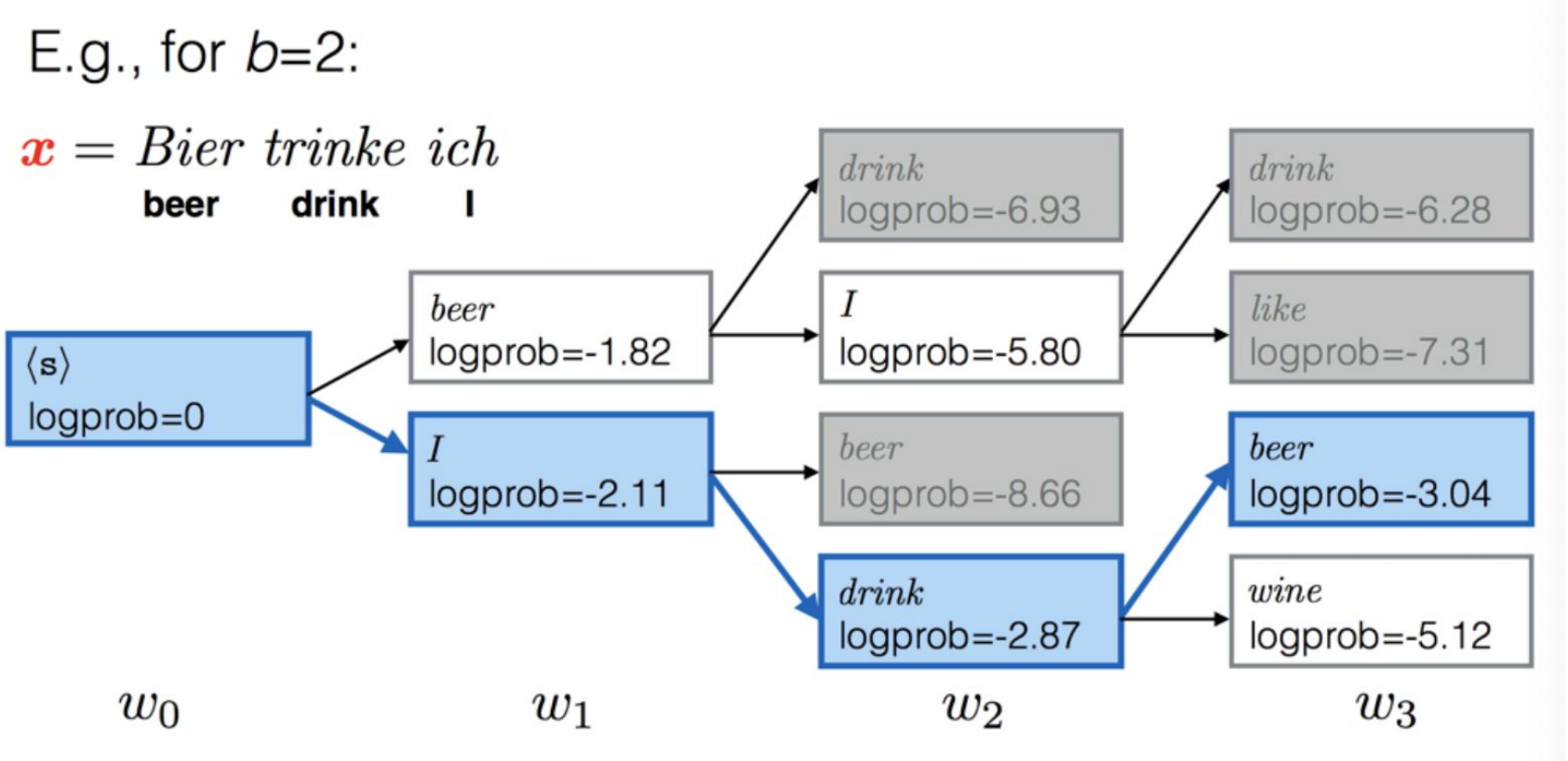
beer **drink** **I**



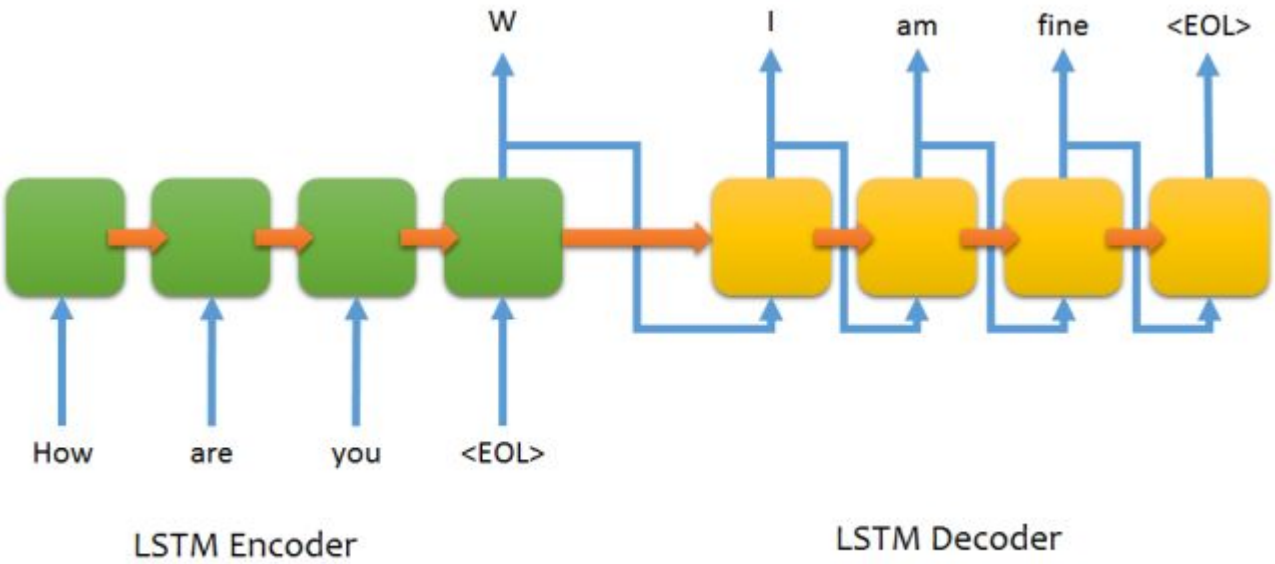
RNN Architectures: Beam Search



RNN Architectures: Beam Search

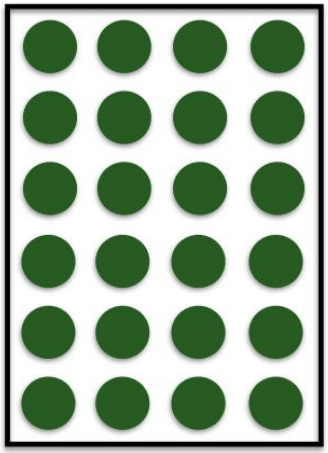


RNN Architectures: Sequence to Sequence

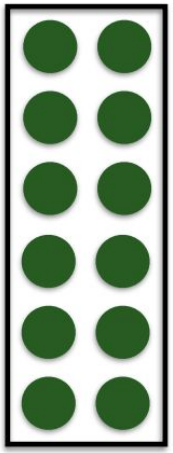


Source

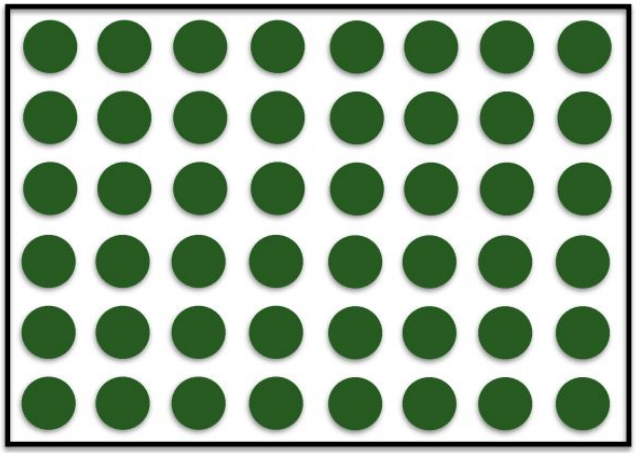
Sentences as matrices



Ich möchte ein Bier



Mach's gut

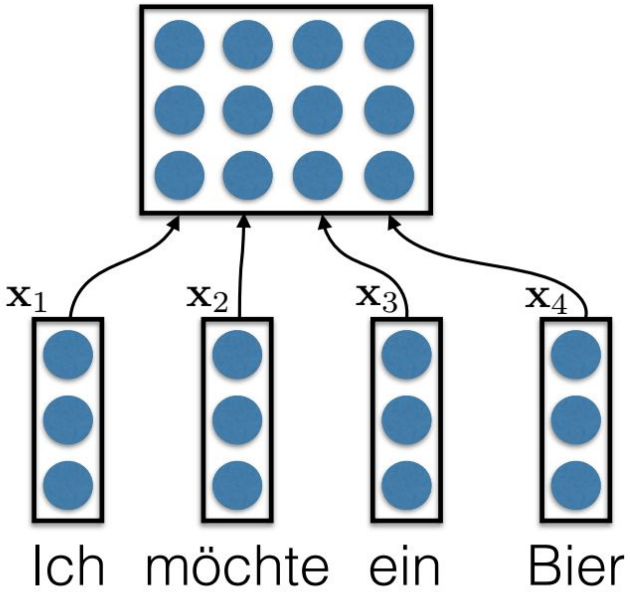


Die Wahrheiten der Menschen sind die unwiderlegbaren Irrtümer

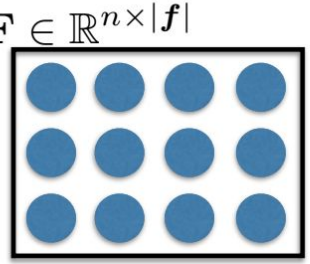
Question: How do we build these matrices?

Sentences as matrices

$$\mathbf{f}_i = \mathbf{x}_i$$

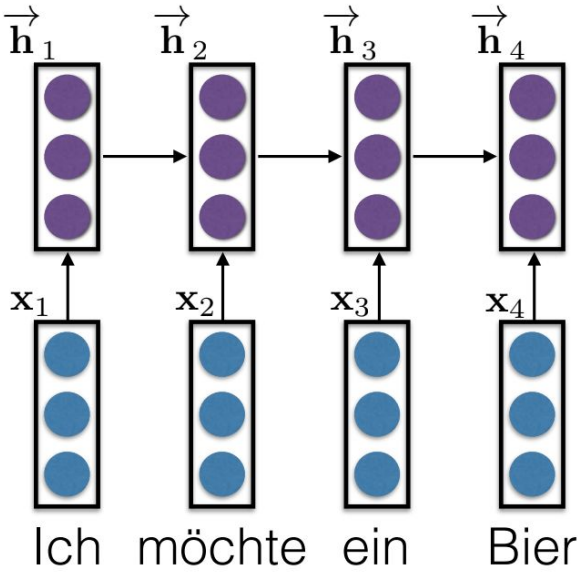


$\mathbf{F} \in \mathbb{R}^{n \times |\boldsymbol{f}|}$

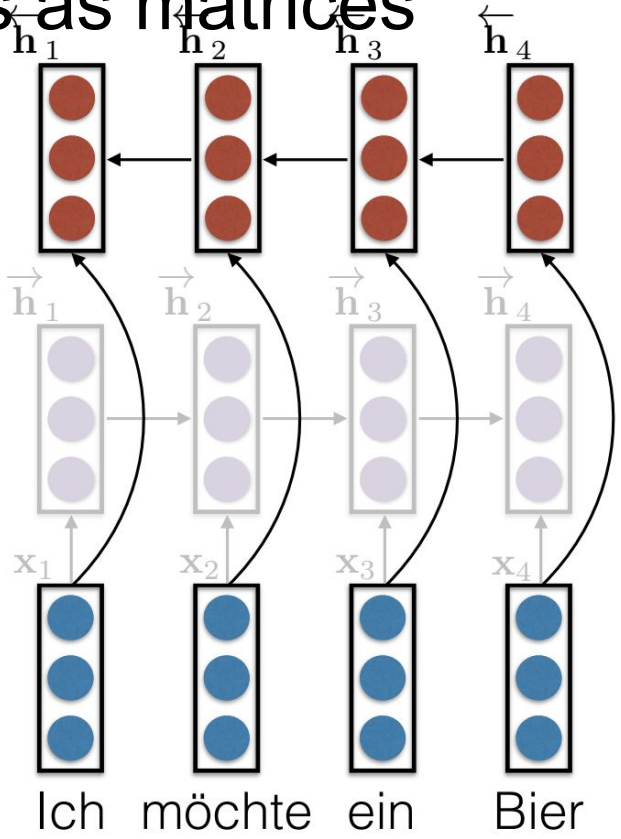


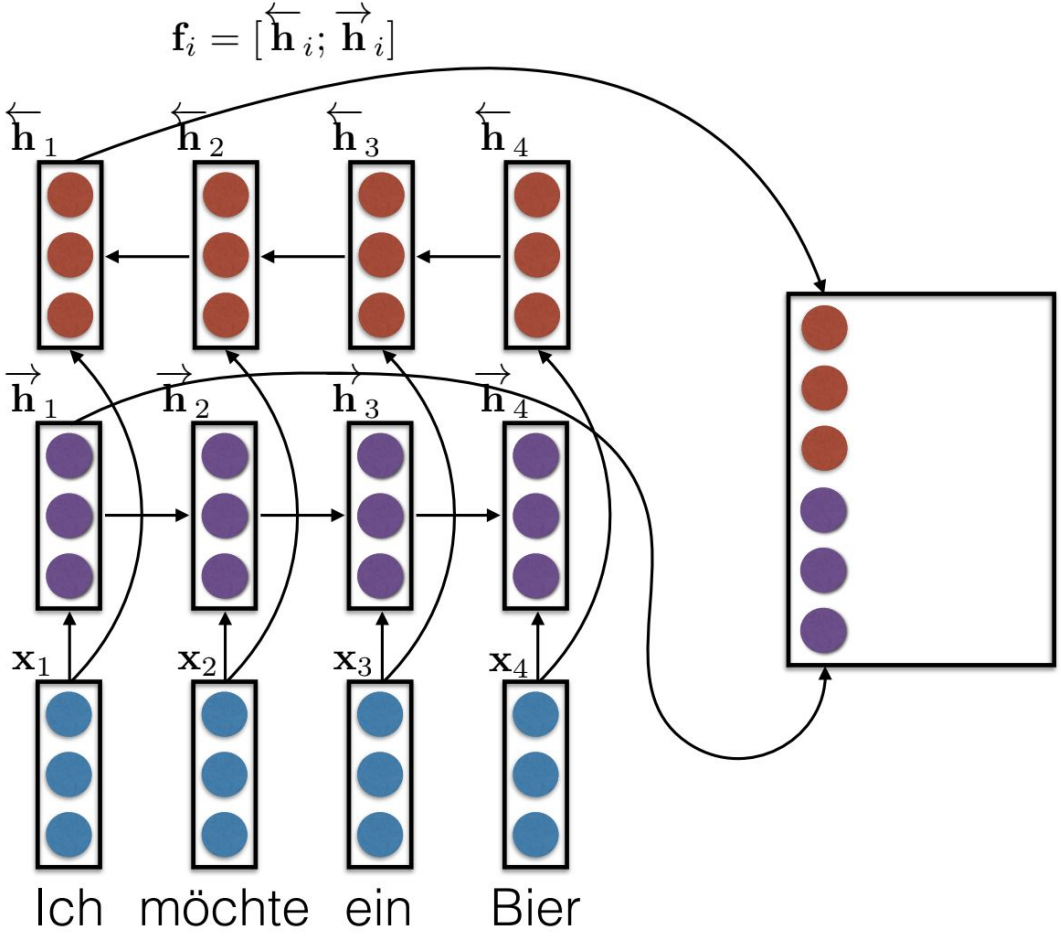
Ich möchte ein Bier

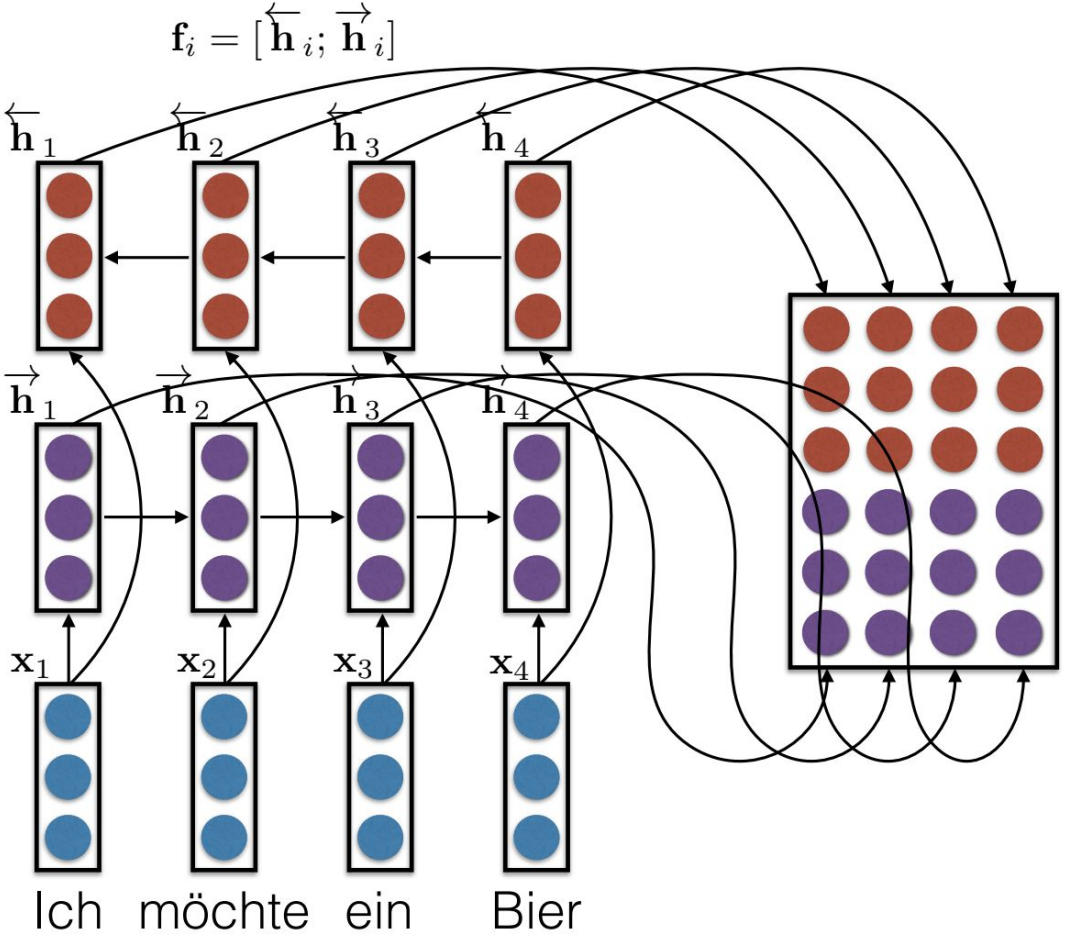
Sentences as matrices



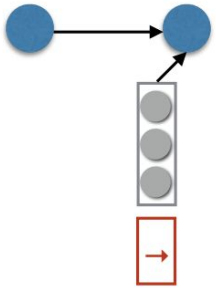
Sentences as matrices



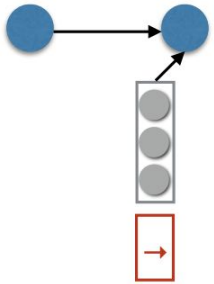

$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$
$\overleftarrow{\mathbf{h}}_1$
$\overleftarrow{\mathbf{h}}_2$
$\overleftarrow{\mathbf{h}}_3$
$\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier

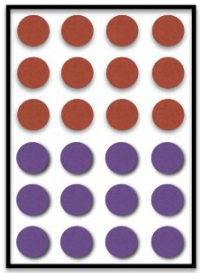

$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$
$\overleftarrow{\mathbf{h}}_1$
$\overleftarrow{\mathbf{h}}_2$
$\overleftarrow{\mathbf{h}}_3$
$\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier

Attention



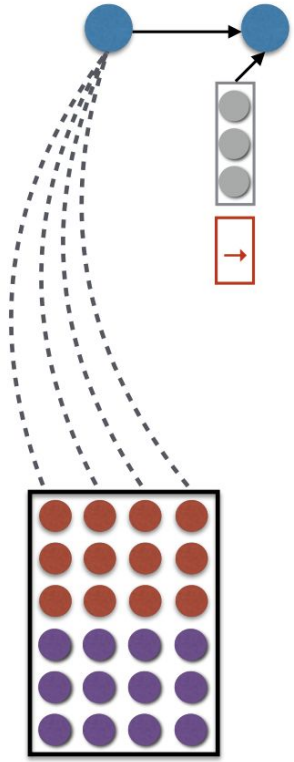
Attention



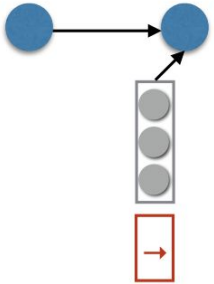


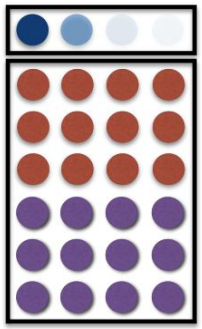
Ich möchte ein Bier

Attention



Attention



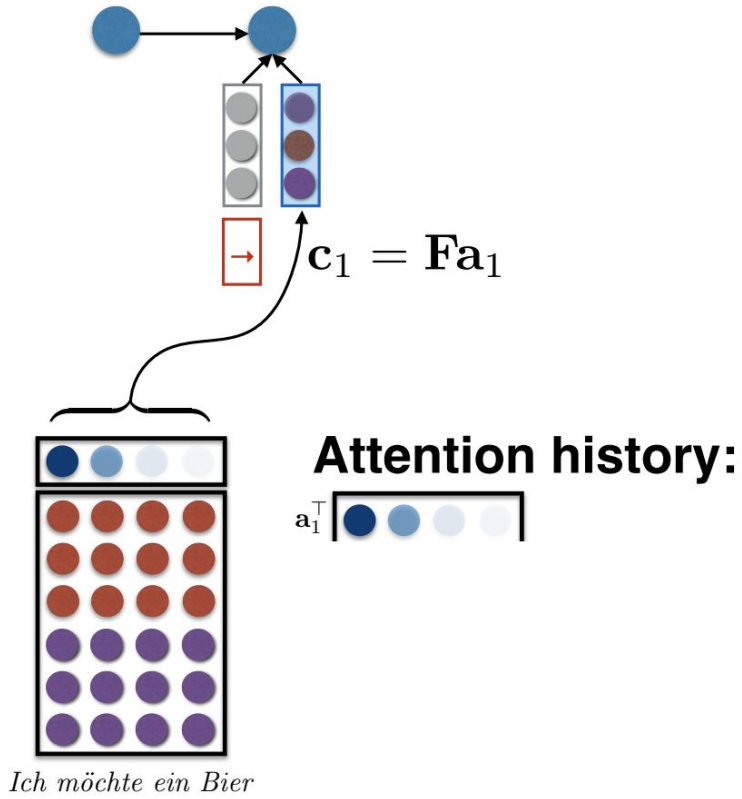


Ich möchte ein Bier

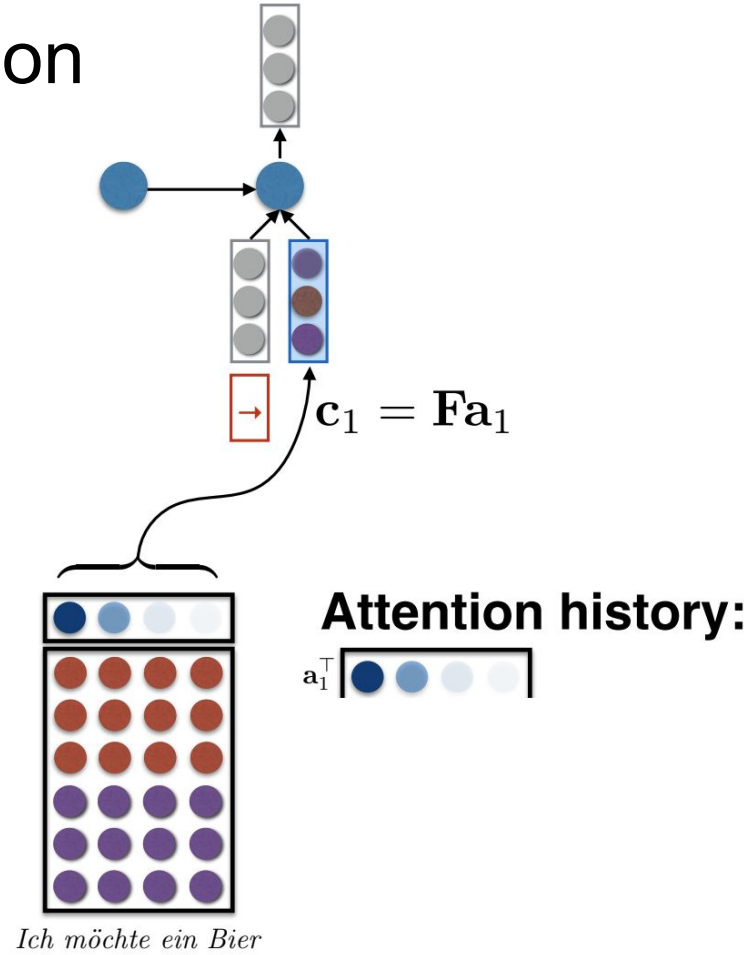
Attention history:

$\mathbf{a}_1^\top$

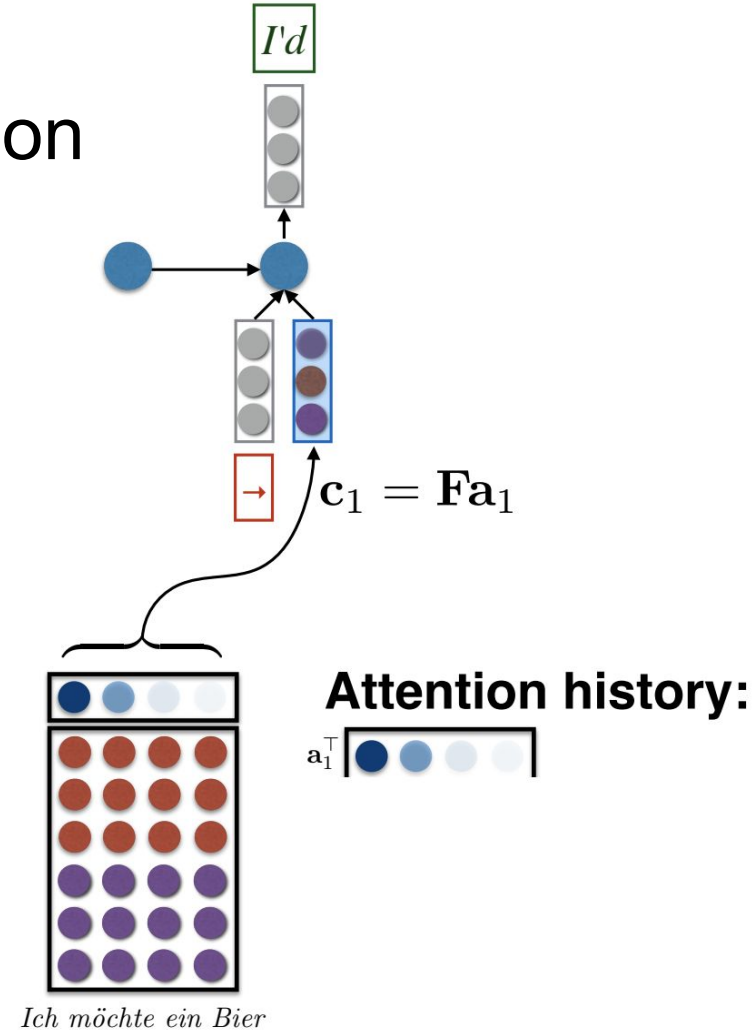
Attention


$\mathbf{c}_1 = \mathbf{F}\mathbf{a}_1$
Attention history:
$\mathbf{a}_1^\top$
Ich möchte ein Bier

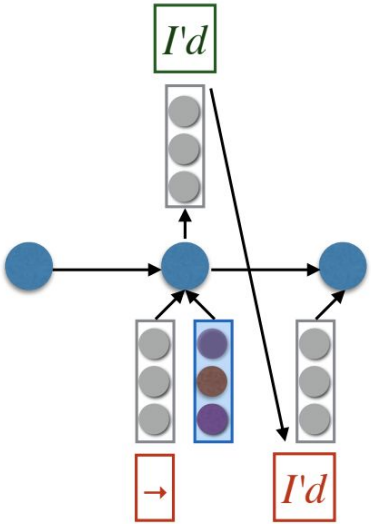
Attention

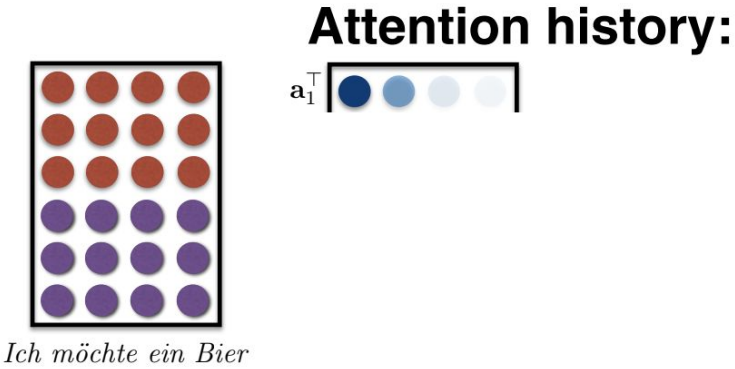

$\mathbf{c}_1 = \mathbf{F}\mathbf{a}_1$
Attention history:
$\mathbf{a}_1^\top$
Ich möchte ein Bier

Attention

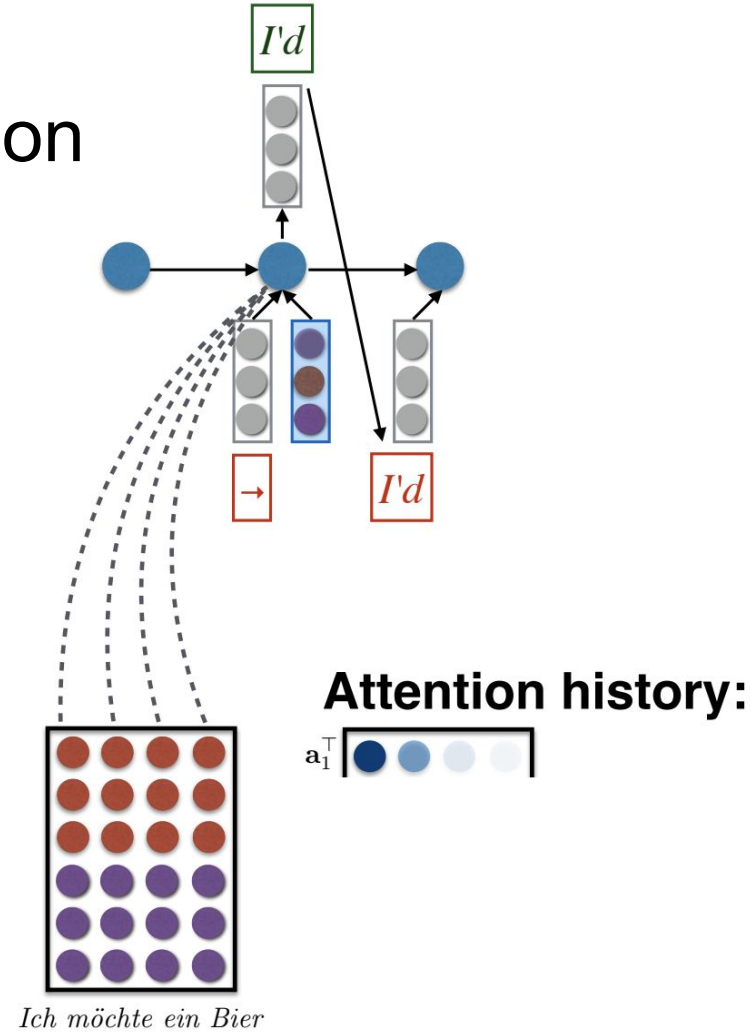

I'd
→
$\mathbf{c}_1 = \mathbf{F}\mathbf{a}_1$
Attention history:
$\mathbf{a}_1^\top$
Ich möchte ein Bier

Attention

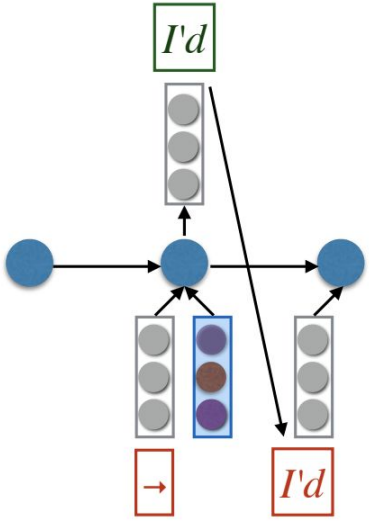


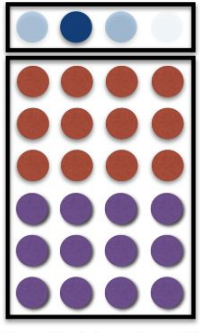


Attention



Attention

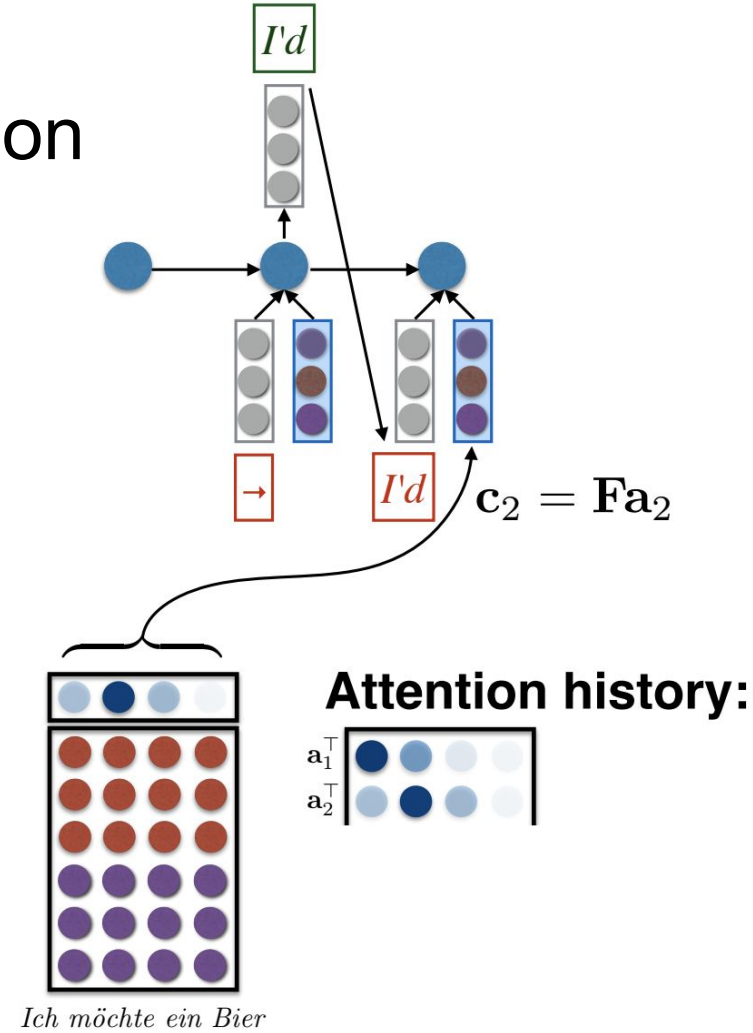




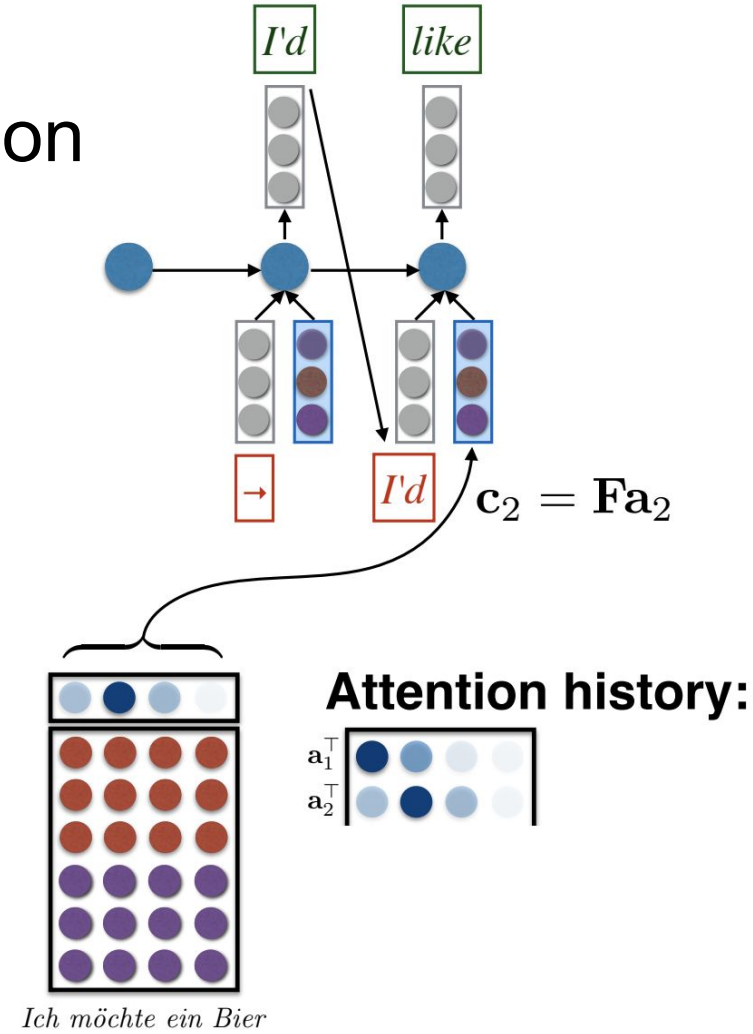
Ich möchte ein Bier

Attention history:

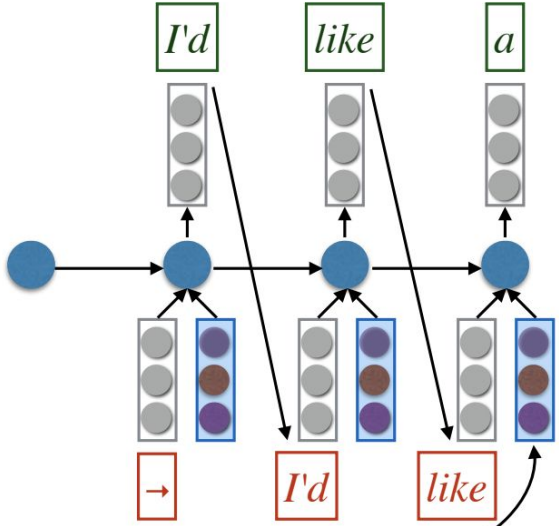
Attention

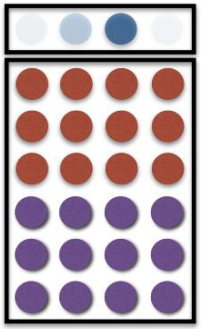

I'd
→
I'd
$\mathbf{c}_2 = \mathbf{F}\mathbf{a}_2$
Attention history:
$\mathbf{a}_1^\top$
$\mathbf{a}_2^\top$
Ich möchte ein Bier

Attention


I'd
like
→
I'd
$\mathbf{c}_2 = \mathbf{F}\mathbf{a}_2$
Attention history:
$\mathbf{a}_1^\top$
$\mathbf{a}_2^\top$
Ich möchte ein Bier

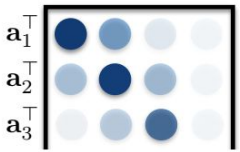
Attention



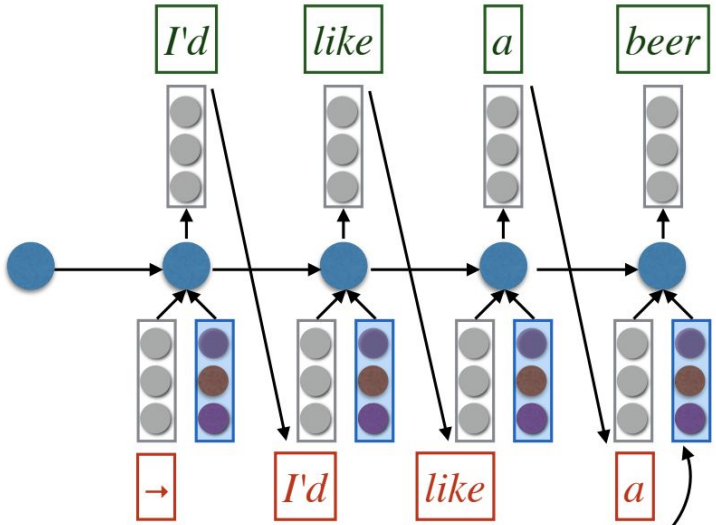


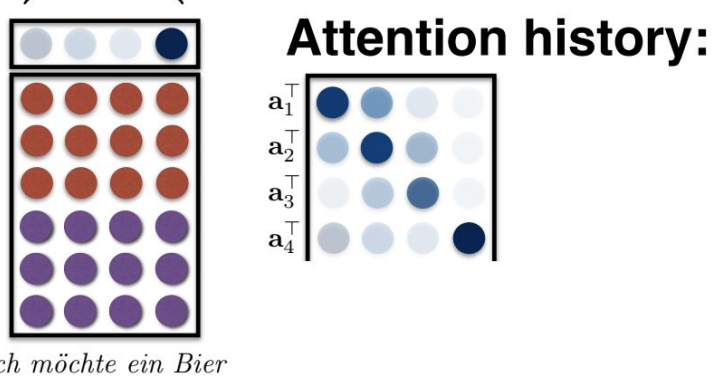
Ich möchte ein Bier

Attention history:



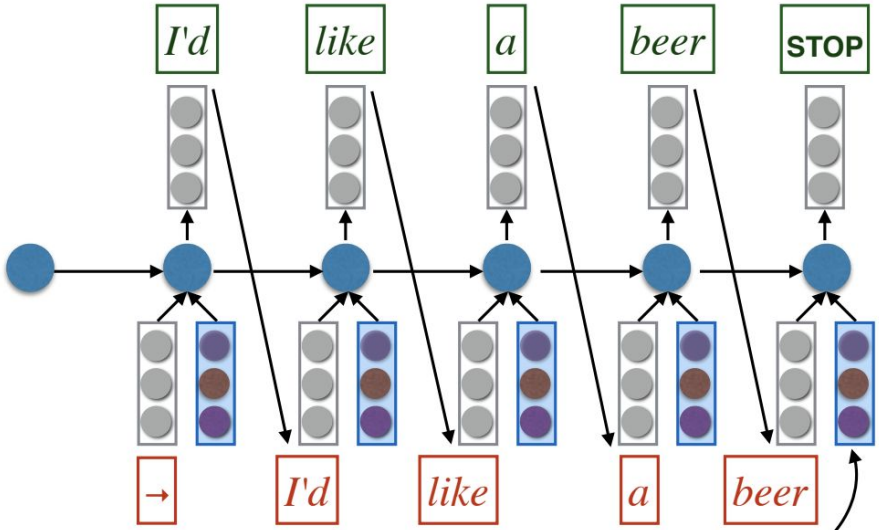
Attention

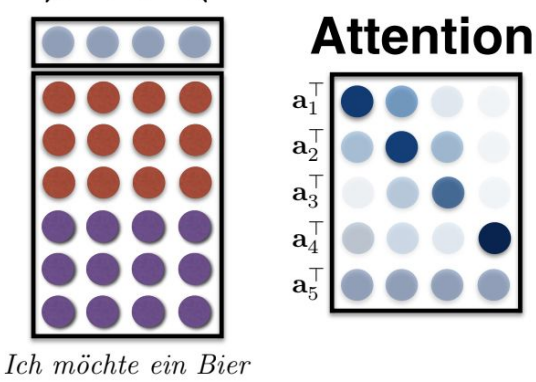




Attention history:

Attention





Attention history:

Outline

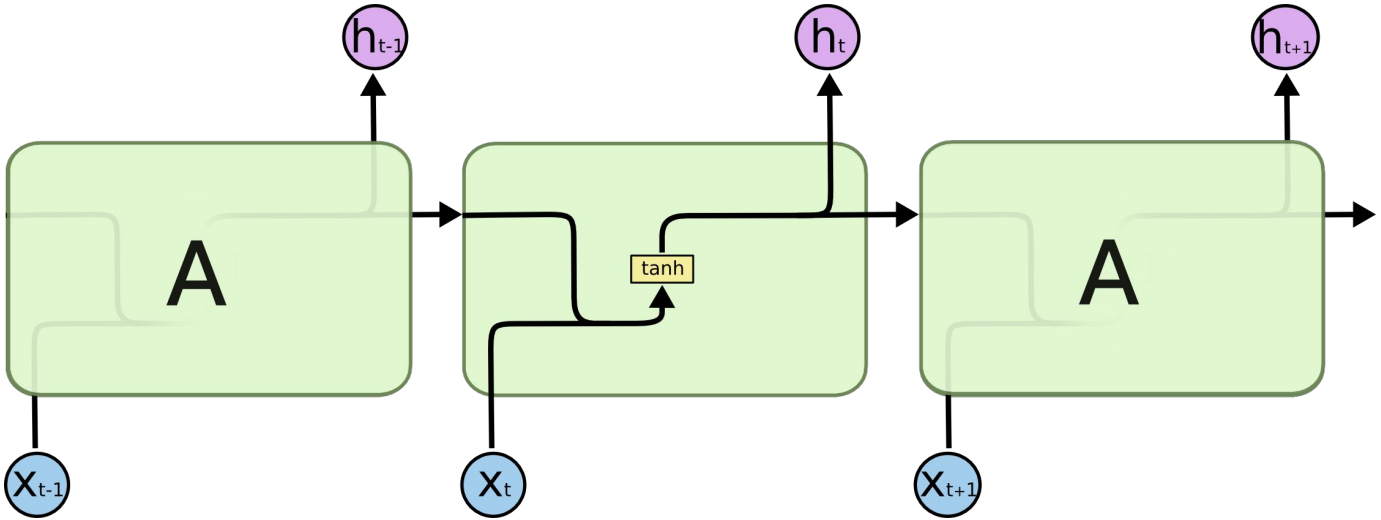
Motivation: Why Sequence Models?

Recurrent Neural Networks: Overview

Recurrent Neural Network Architectures

Types of RNN Cells

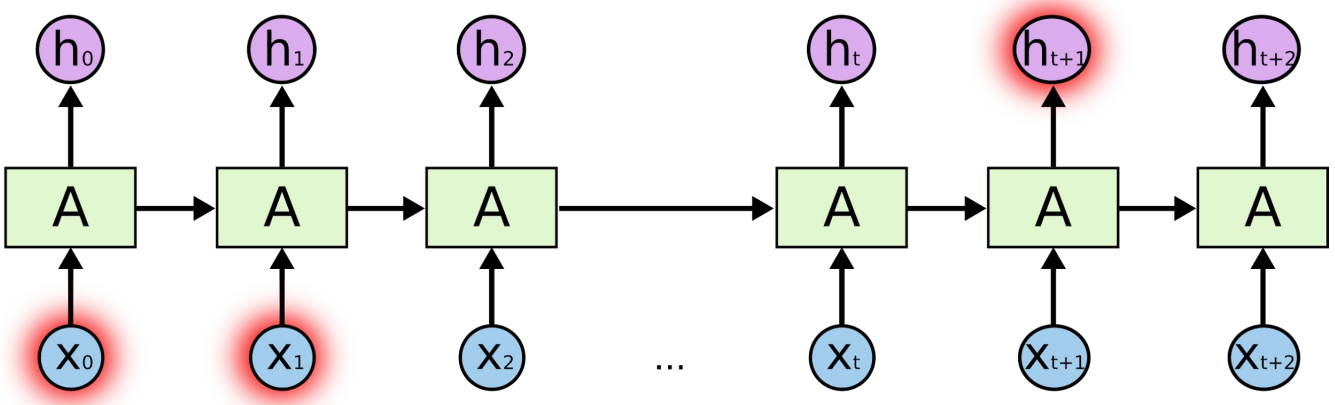
Simple RNN Cell Example



$$\mathrm{h}_t = \tanh\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

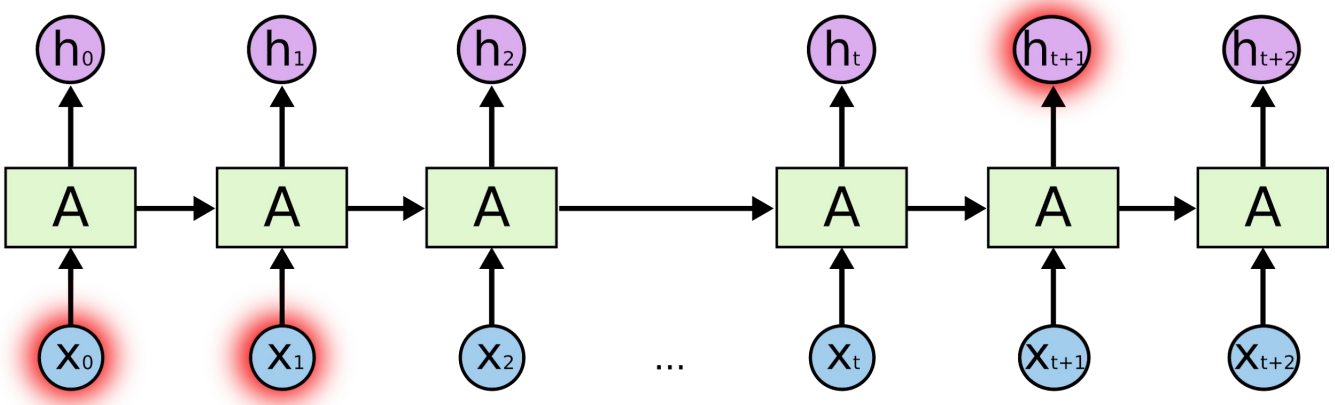
Taken from http://colah.github.io/posts/2015-08-Understanding-LSTMs/ [3].

Problem: Vanishing Gradients



Taken from [3].

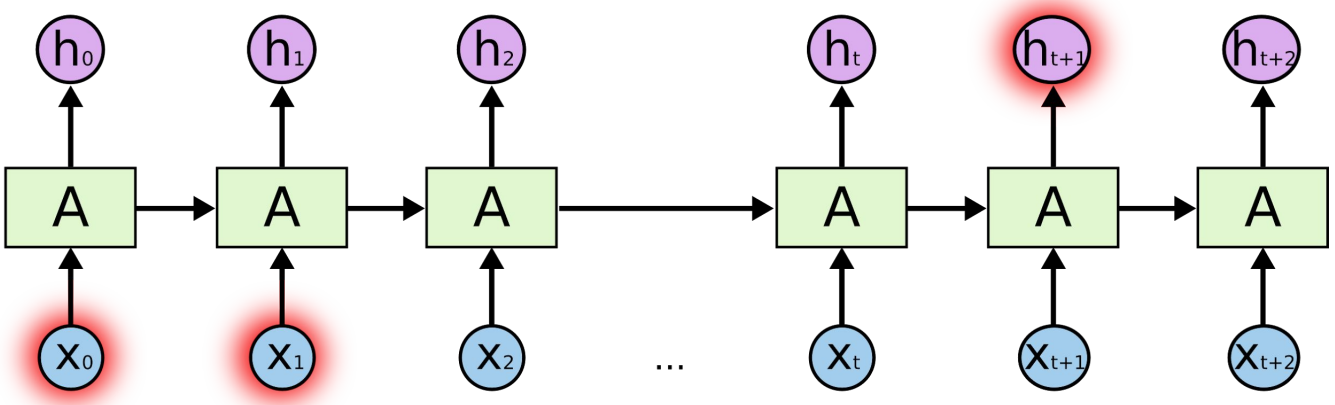
Problem: Vanishing Gradients



Using simple RNN cells, the model has difficulty learning connections between inputs and outputs that are far apart.

Taken from [3].

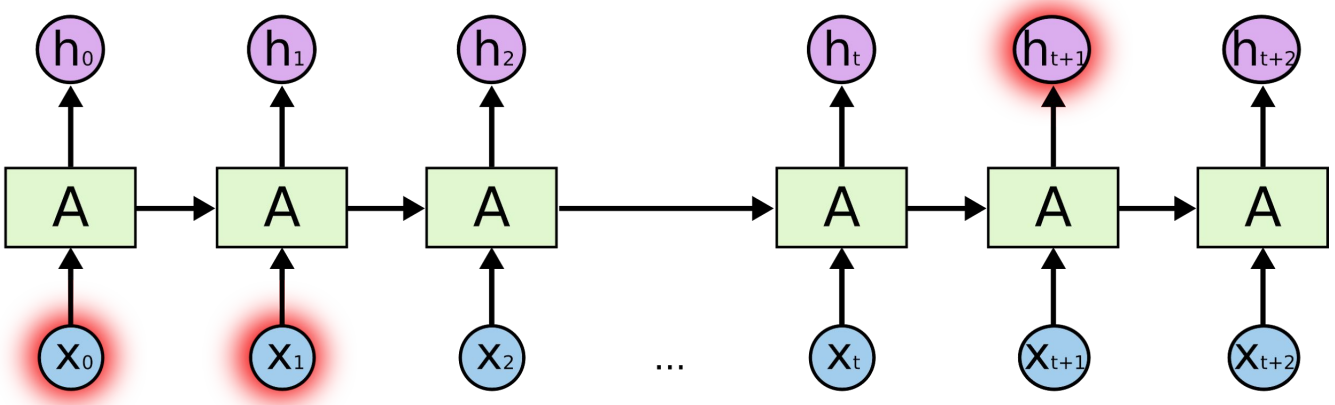
Problem: Vanishing Gradients



In other words, your model has very small gradients.

Taken from [3].

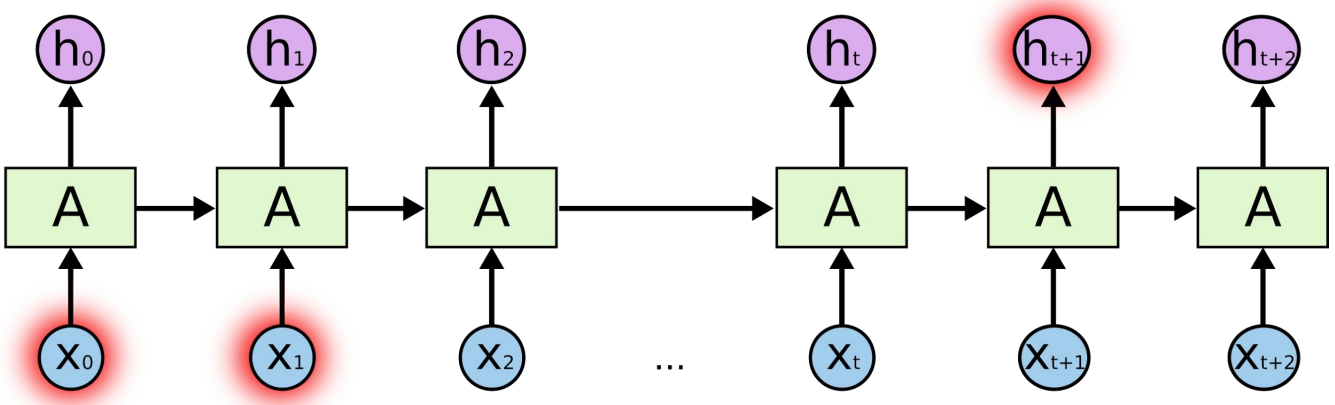
Problem: Vanishing Gradients



Gradients from *t+T* at time *t* vanish exponentially with *T* **if at each time step |dh(t)/dh(t-1)| < 1.0**

Taken from [3].

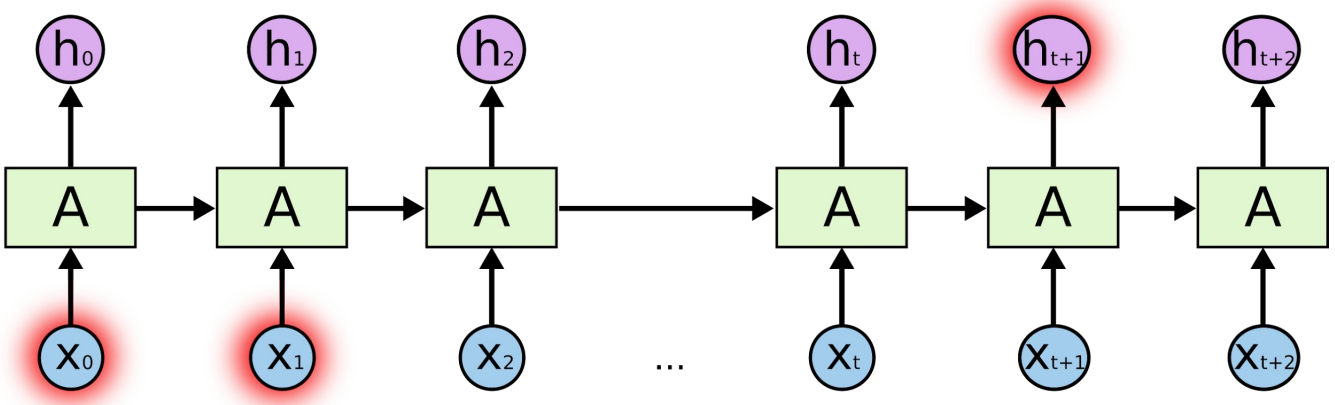
Problem: Vanishing Gradients



In other words, your model has very small gradients.
It will learn extremely slowly.

Taken from [3].

Problem: Gradients Exploding

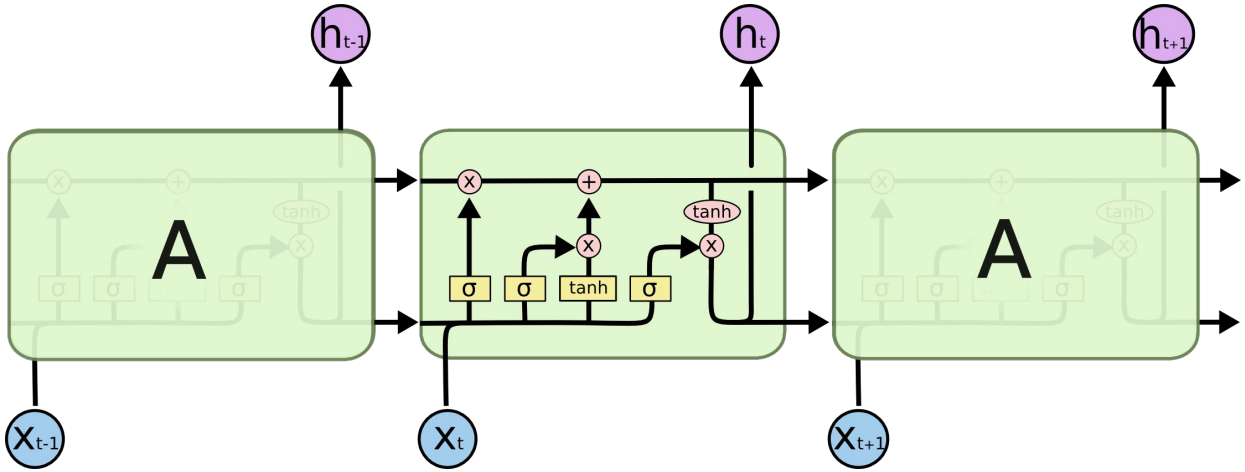


Opposite issue! Start to see NaNs during training

Taken from [3].

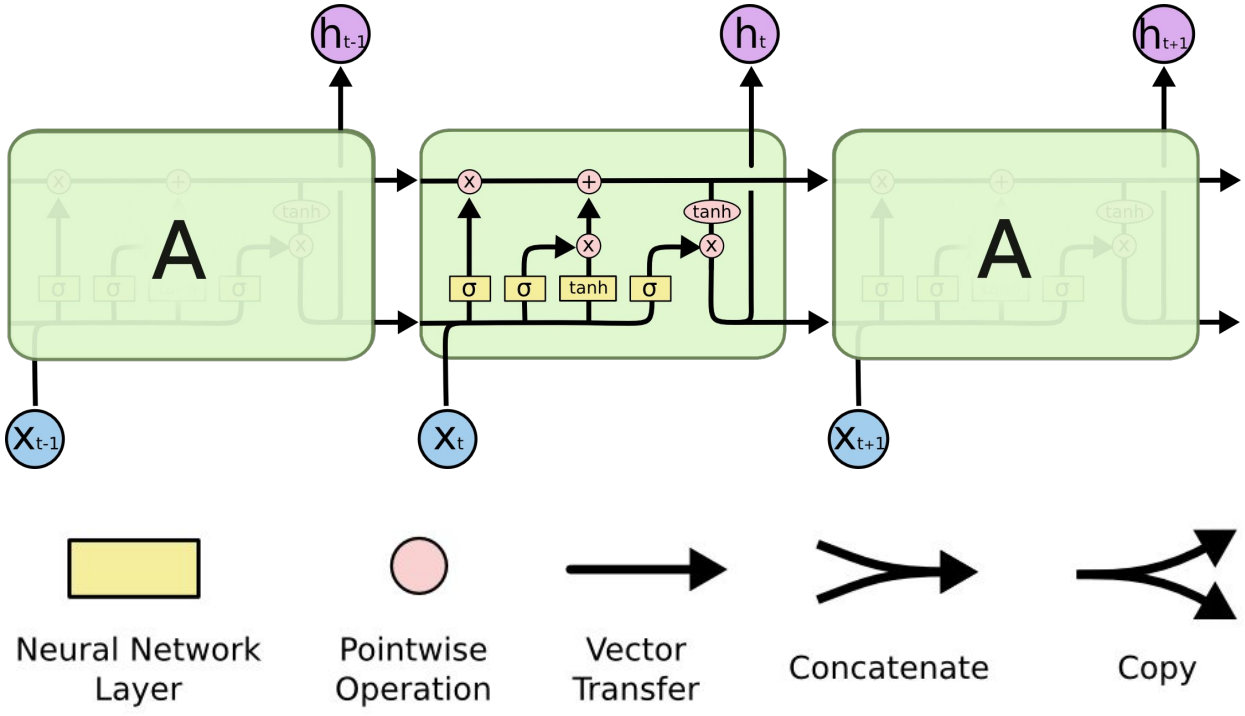
Reference: On the difficulty of training recurrent neural networks

A solution: Long Short Term Memory Cells



Taken from [3].

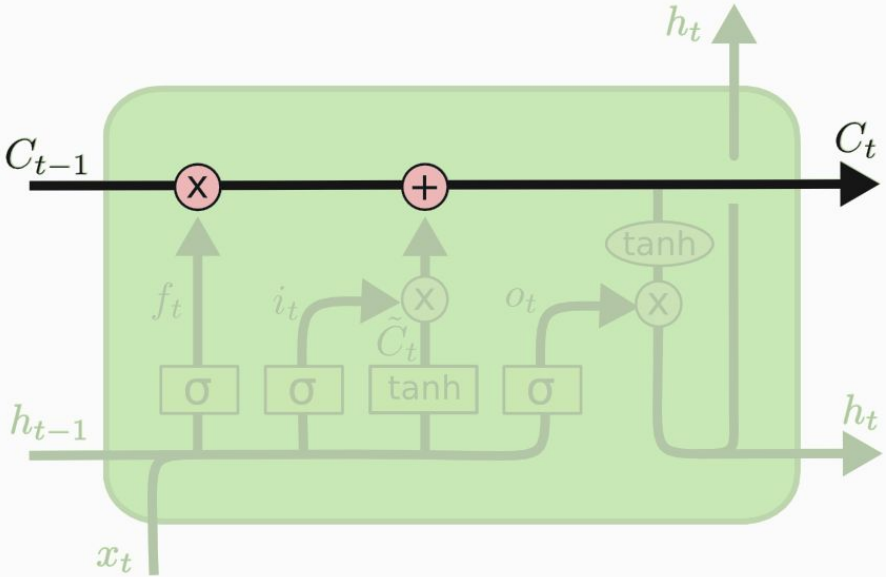
A solution: Long Short Term Memory Cells



Taken from [3].

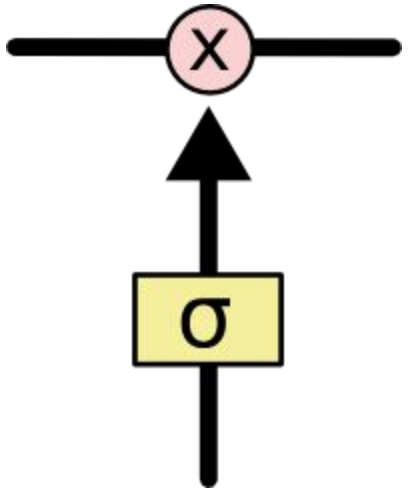
LSTM - Cell State

- “Conveyer belt”
- LSTM can “add” or “remove” information to cell state via *gates*.



Taken from [3].

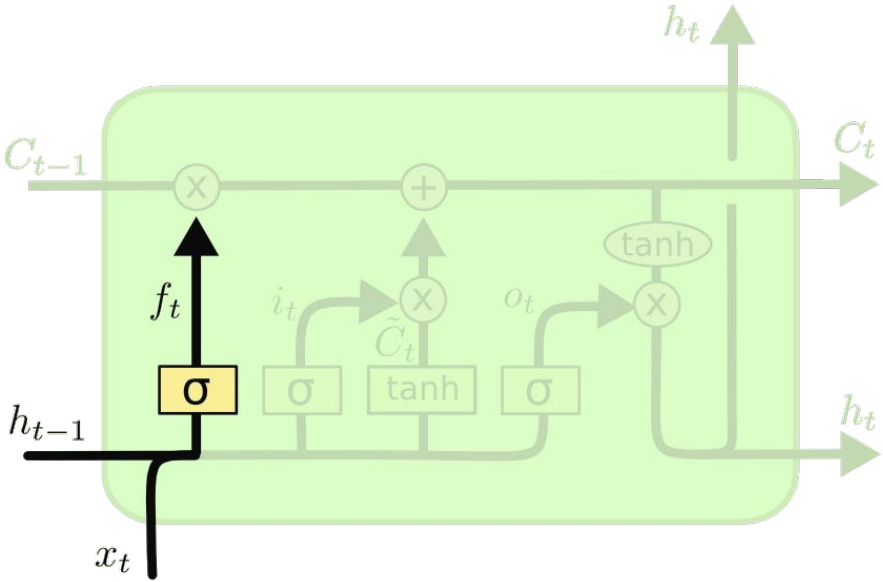
Gates: Optionally Let Information Through



- Element-wise sigmoid and element-wise multiplication
- Differentiable: trainable
- $\sigma(\cdot) \in [0, 1]$

Taken from [3].

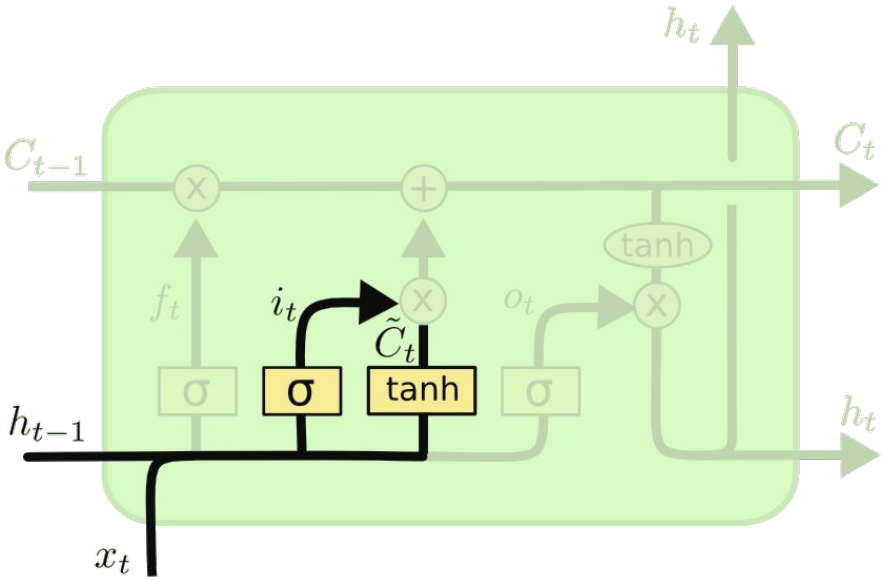
LSTM Cell: Forget Gate



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Taken from [3].

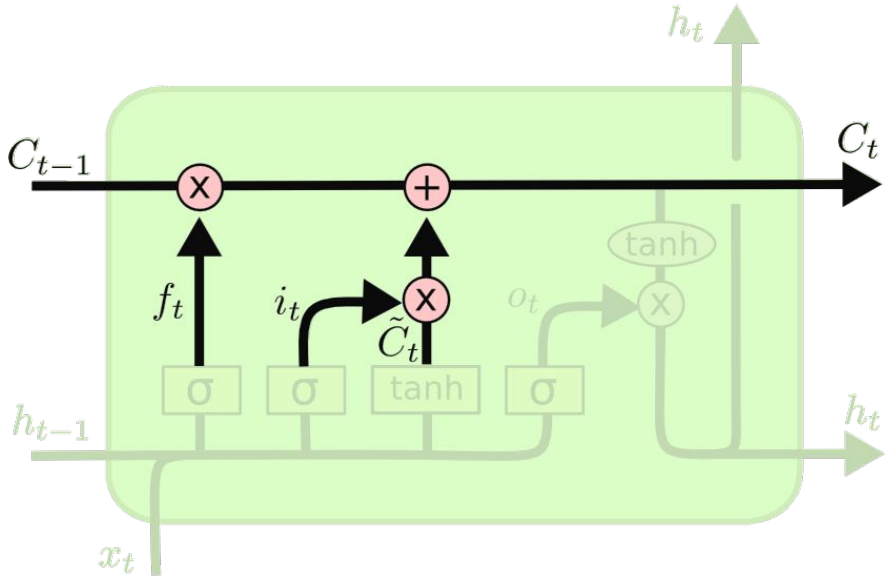
LSTM Cell: Input Gate and Candidate State



?

Taken from [3].

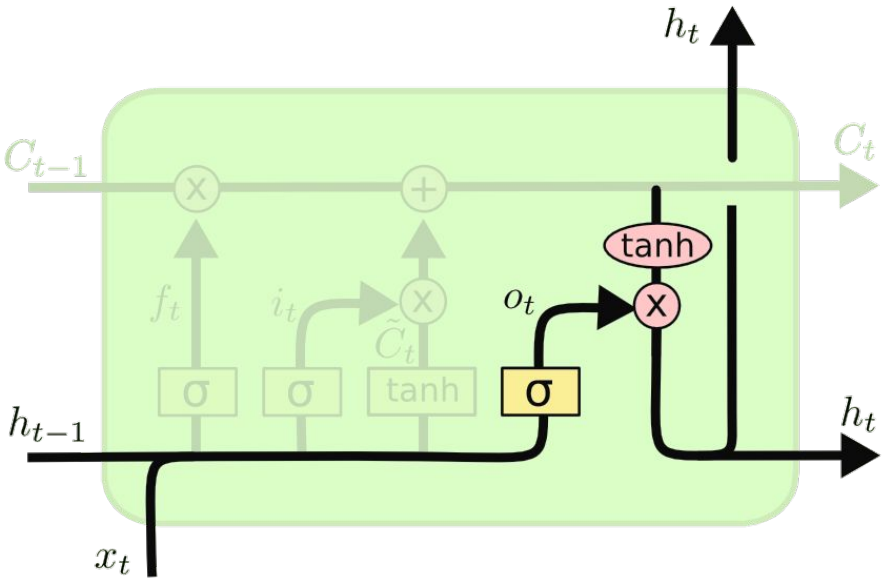
LSTM Cell: Update the Cell State



?

Taken from [3].

LSTM Cell: Output Gate



?

- Output filtered version of cell state
- $\tanh(\cdot) \in [-1, 1]$

Taken from [3].

LSTM Cell: Take Away

LSTM cells propagate state on a '**conveyor belt**.'

LSTM cells contain a series of **gates** that alter state at each time step.

Some gates **add** to the state while other gates **forget**.

LSTM cells can propagate information over many time steps.

They can also erase state quickly.

Taken from [3].

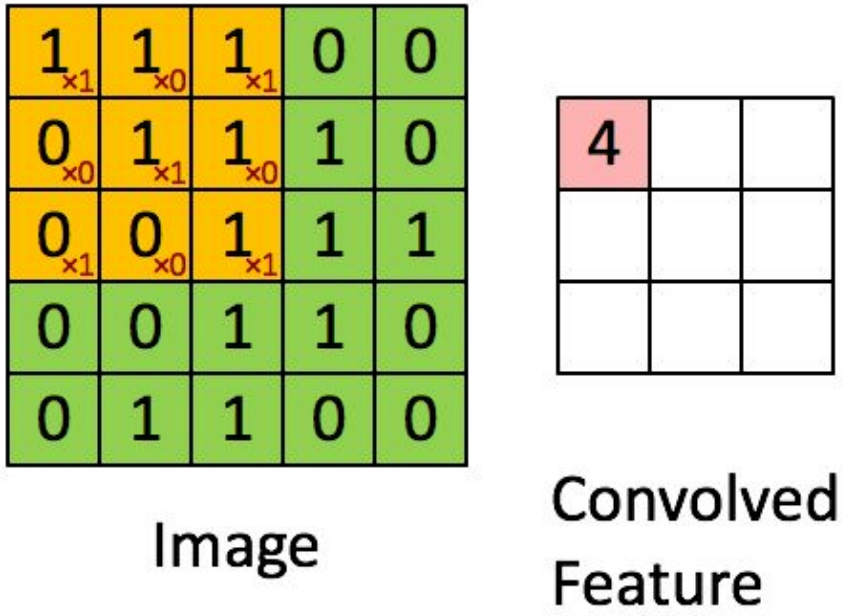
Empirical results

- LSTM is probably the most popular, but:
- People use different variations of LSTM, gated recurrent units (GRU), etc.

RNN Cells In TensorFlow

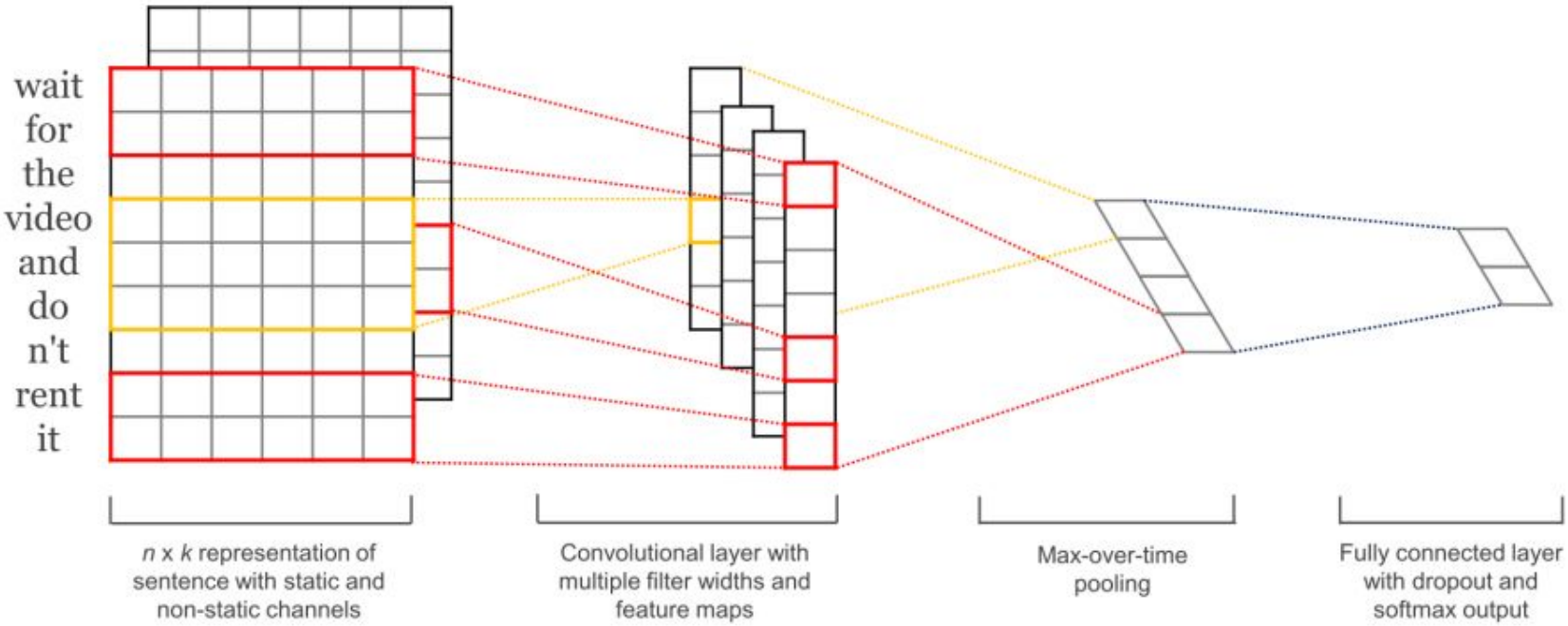
https://www.tensorflow.org/tutorials/recurrent

Convolutional Neural Networks



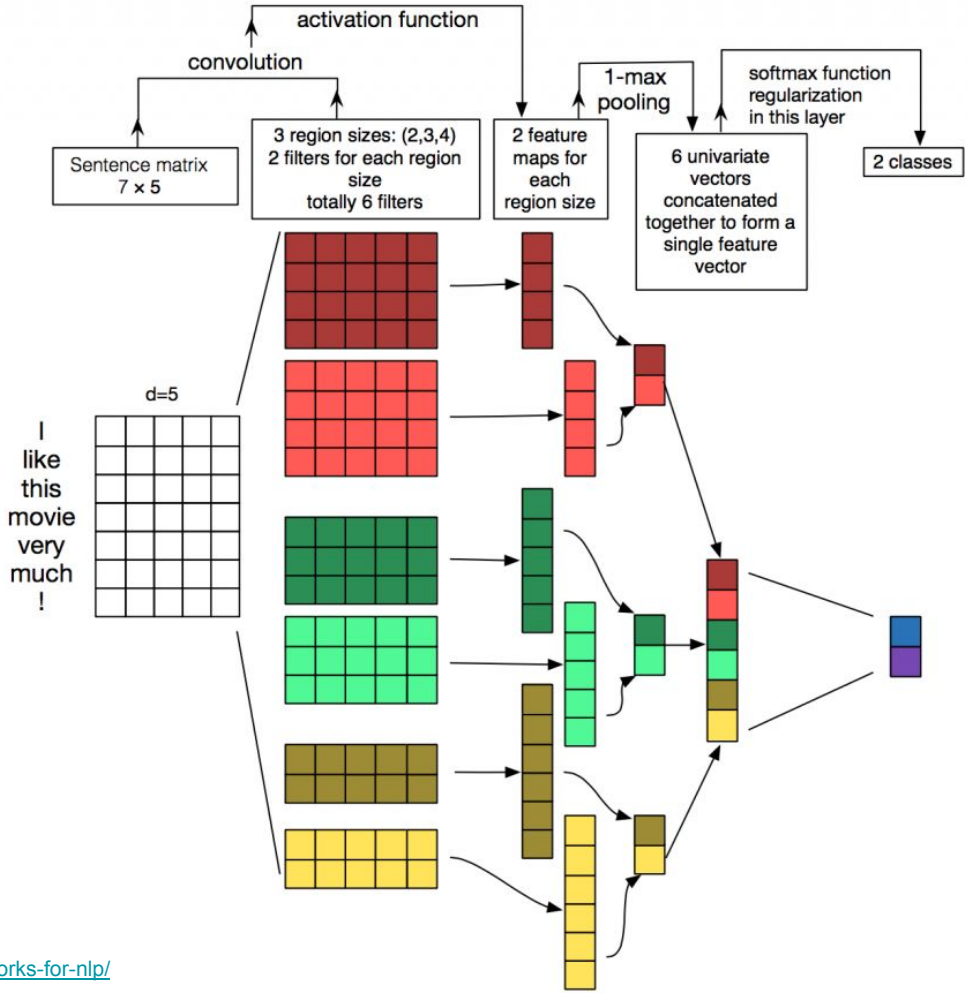
http://deeplearning.stanford.edu/wiki/images/6/6c/Convolution_schematic.gif

CNN



Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification

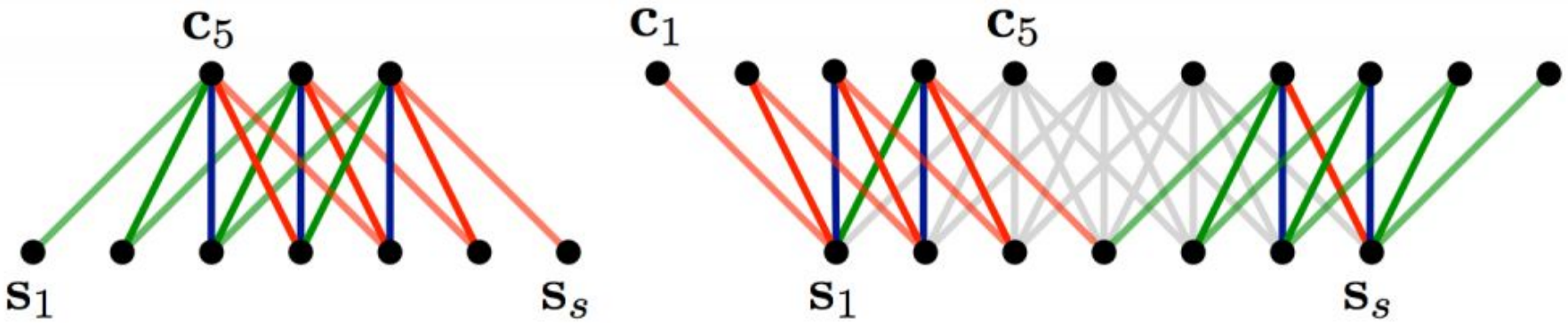
Convolutional Neural Networks on Text



Source: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Convolutional Neural Networks on Text

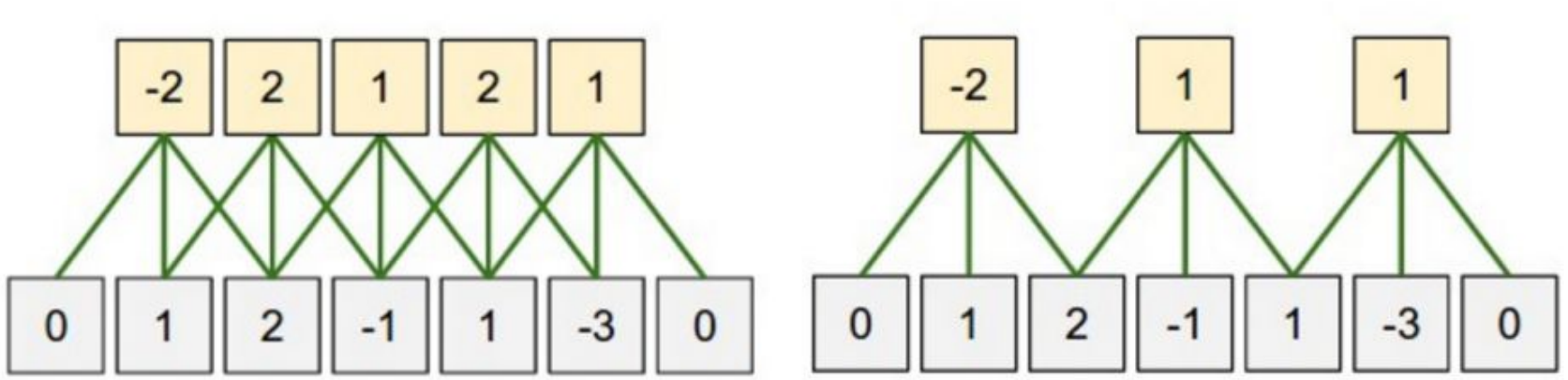
Narrow vs Wide Convolution



Narrow vs. Wide Convolution. Filter size 5, input size 7. Source: A Convolutional Neural Network for Modelling Sentences (2014)

Convolutional Neural Networks on Text

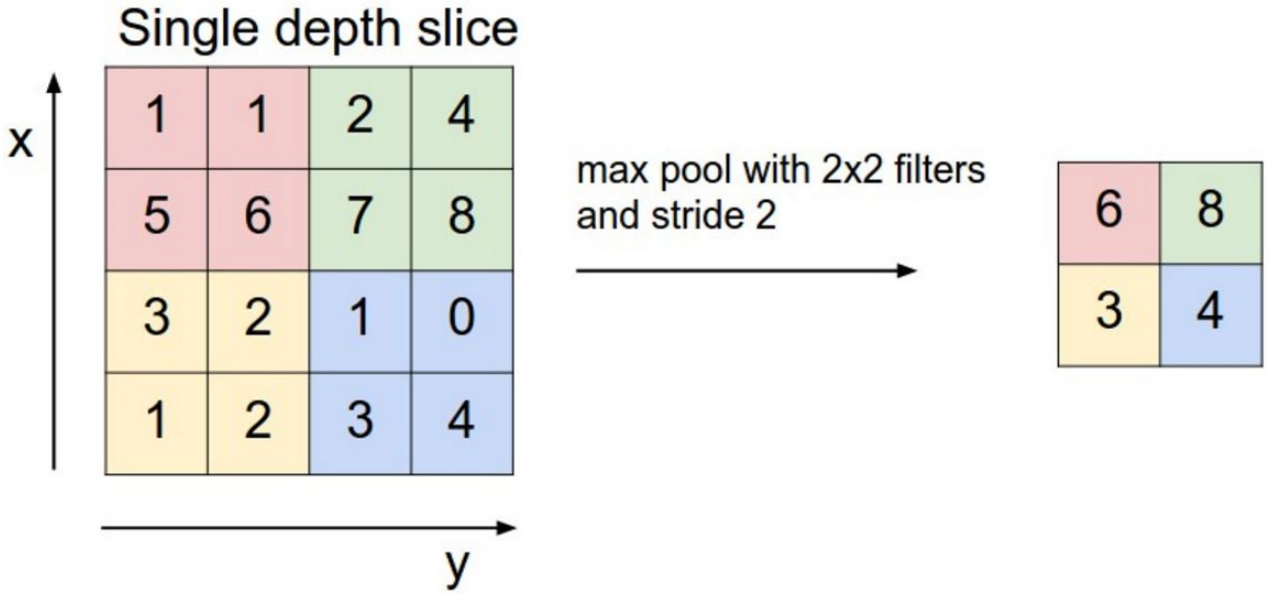
Stride Size



Convolution Stride Size. Left: Stride size 1. Right: Stride size 2. Source: http://cs231n.github.io/convolutional-networks/

Convolutional Neural Networks on Text

Pooling Layers



Max pooling in CNN. Source: http://cs231n.github.io/convolutional-networks/#pool

Convolutional Neural Networks on Text

Channels

RGB for images

Word2Vec, Glove, etc for text

Case Studies

SmartReply

Text Simplification and its evaluation

All the best!

Feel free to ask me anything related to Google, NLP, ML, DL, PhD, Career, Online Activism, Sport :)

me@juliapro.xyz