Natural Language Processing

Lecture : Part-of-Speech Tagging

Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Part-of-speech tagging

©The New York Times

Part-of-speech

©seo-herd

Part-of-speech

Part-of-speech (POS) tags are lexical categories such as noun, verb, adjective, adverb, pronoun, preposition, article, etc.

Also known as word classes or morphological classes. Recall these classes are defined either distributionally or else functionally (see essentials of linguistics lecture).

We call **tagset** the set of all POS tags used by some model.

Different languages, different grammatical theories, and different applications may require different tagsets.

Part-of-speech

POS tags fall into two broad categories: closed class and open class.

Closed class includes prepositions, pronouns, articles, etc. New words in this class are rarely coined.

Open class consists of four major groups: nouns (including proper nouns), verbs, adjectives, and adverbs. New words appear almost always in this class.

Interjections are also a (minor) group in open class.

Part-of-speech

The Universal Dependencies (UD) tagset contains 17 tags:

	Tag	Description	Example
Open Class	**ADJ**	Adjective: noun modifiers describing properties	*red, young, awesome*
	ADV	Adverb: verb modifiers of time, place, manner	*very, slowly, home, yesterday*
	NOUN	words for persons, places, things, etc.	*algorithm, cat, mango, beauty*
	VERB	words for actions and processes	*draw, provide, go*
	PROPN	Proper noun: name of a person, organization, place, etc..	*Regina, IBM, Colorado*
	INTJ	Interjection: exclamation, greeting, yes/no response, etc.	*oh, um, yes, hello*
Closed Class Words	**ADP**	Adposition (Preposition/Postposition): marks a noun's spacial, temporal, or other relation	*in, on, by under*
	AUX	Auxiliary: helping verb marking tense, aspect, mood, etc.,	*can, may, should, are*
	CCONJ	Coordinating Conjunction: joins two phrases/clauses	*and, or, but*
	DET	Determiner: marks noun phrase properties	*a, an, the, this*
	NUM	Numeral	*one, two, first, second*
	PART	Particle: a preposition-like form used together with a verb	*up, down, on, off, in, out, at, by*
	PRON	Pronoun: a shorthand for referring to an entity or event	*she, who, I, others*
	SCONJ	Subordinating Conjunction: joins a main clause with a subordinate clause such as a sentential complement	*that, which*
Other	**PUNCT**	Punctuation	; , ()
	SYM	Symbols like $ or emoji	$, %
	X	Other	asdf, qwfg

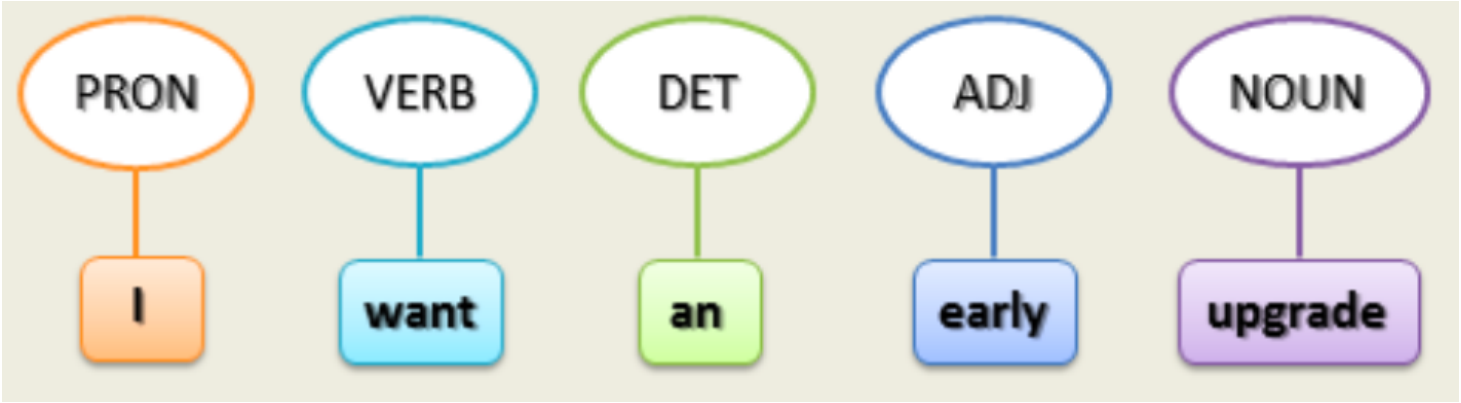
UD dataset has POS tagged corpora for 100+ languages, at time of writing.

Part-of-speech

The English-specific Penn Treebank (PTB) tagset is also very popular; it contains 45 tags.

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coord. conj.	*and, but, or*	NNP	proper noun, sing.	*IBM*	TO	"to"	*to*
CD	cardinal number	*one, two*	NNPS	proper noun, plu.	*Carolinas*	UH	interjection	*ah, oops*
DT	determiner	*a, the*	NNS	noun, plural	*llamas*	VB	verb base	*eat*
EX	existential 'there'	*there*	PDT	predeterminer	*all, both*	VBD	verb past tense	*ate*
FW	foreign word	*mea culpa*	POS	possessive ending	*'s*	VBG	verb gerund	*eating*
IN	preposition/ subordin-conj	*of, in, by*	PRP	personal pronoun	*I, you, he*	VBN	verb past participle	*eaten*
JJ	adjective	*yellow*	PRP$	possess. pronoun	*your, one's*	VBP	verb non-3sg-pr	*eat*
JJR	comparative adj	*bigger*	RB	adverb	*quickly*	VBZ	verb 3sg pres	*eats*
JJS	superlative adj	*wildest*	RBR	comparative adv	*faster*	WDT	wh-determ.	*which, that*
LS	list item marker	*1, 2, One*	RBS	superlatv. adv	*fastest*	WP	wh-pronoun	*what, who*
MD	modal	*can, should*	RP	particle	*up, off*	WP$	wh-possess.	*whose*
NN	sing or mass noun	*llama*	SYM	symbol	+,%, &	WRB	wh-adverb	*how, where*

Part-of-speech tagging



Part-of-speech tagging

Words are **ambiguous**: depending on the context in which they appear, they may have different tags.

Example : Word 'book' can be tagged as VERB or as NOUN

- **book/VERB** that flight
- hand me that **book/NOUN**

Only about 15% of the word types in the Brown corpus are ambiguous. But 67% of the word tokens are ambiguous.

Part-of-speech tagging

The task of **part-of-speech tagging** involves the assignment of the **proper** (unique) POS tag to each word in an input sentence.

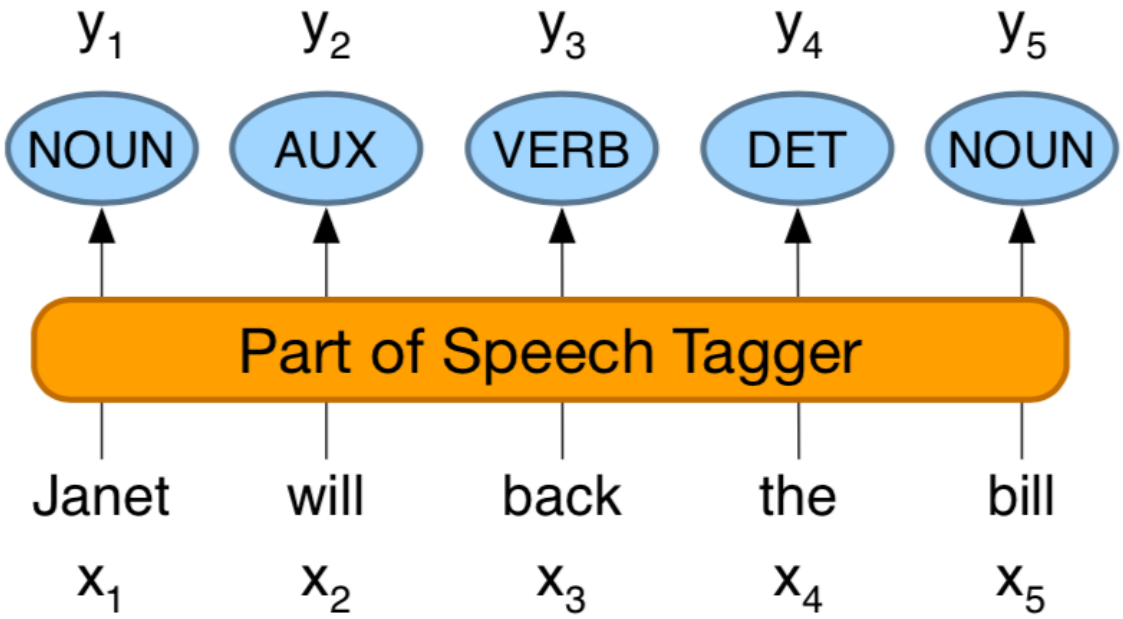
POS tagging must be done in the context of an entire sentence, on the basis of the **grammatical relationship** of a word with its neighboring words.

This is an instance of a more general task called sequence labelling, which we will discuss later.

Part-of-speech tagging

The input is a sequence $x_1, x_2, \ldots, x_n$ of (tokenized) words, and the output is a sequence $y_1, y_2, \ldots, y_n$ of tags, with y_i the tag assigned to x_i.

We assume the tagset is fixed, not part of the input.



Part-of-speech tagging

In POS tagging we need to output a whole sequence of tags $y_1, y_2, \ldots, y_n$ for the input string, not just a category.

POS tagging is therefore a **structured prediction** task, not a classification task.

More details later on structured prediction problems.

The number of output structures can be **exponentially** large in the length of the input, which makes structured prediction more challenging than classification.

Evaluation

Evaluation

The **accuracy** of a part-of-speech tagger is the percentage of test set tags that match human gold labels.

Human ceiling: how often do human annotators agree on the same tag? For PTB this is around 97%.

Accuracies over 97% have been reported across several languages, using the UD tagset.

This also holds for the algorithms we will present in this lecture.

Evaluation

Most Frequent Class baseline: assigning each token to the class it occurred in most often in the training set. This baseline has an accuracy of about 92%.

The most frequent tag NOUN is assigned to unknown words.

Always compare your classifier against a baseline at least as good as the most frequent class baseline.

Evaluation

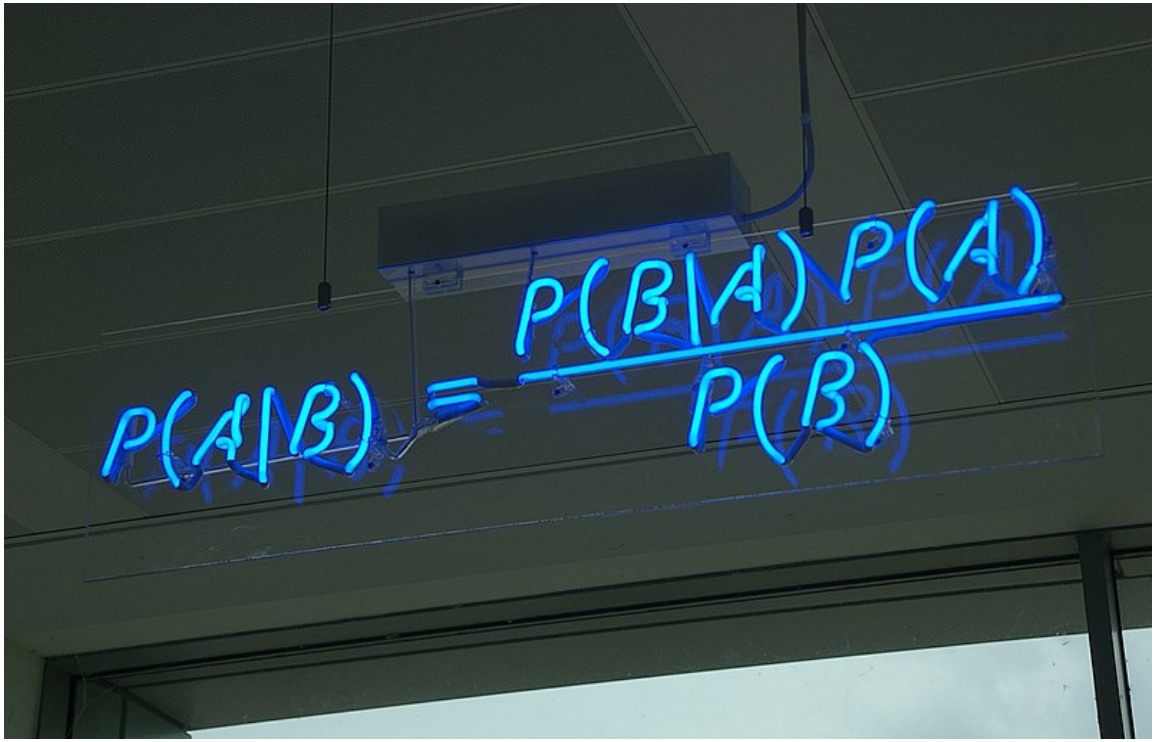
Error analysis: generate a **confusion matrix** for development set. This is a record of how often a word with gold tag y_i was mistagged as y_j, $j \neq i$.

Correct Tags

Predicted Tags	IN	JJ	NN	NNP	RB	VBD	VBN
IN	–	.2			.7		
JJ	.2	–	3.3	2.1	1.7	.2	2.7
NN		8.7	–				.2
NNP	.2	3.3	4.1	–	.2		
RB	2.2	2.0	.5		–		
VBD		.3	.5			–	4.4
VBN		2.8				2.6	–

% of errors caused by mistagging VBN as JJ

Hidden Markov model



https://en.wikipedia.org/wiki/Bayes%27_theorem

Hidden Markov model

Hidden Markov models (HMMs) first applied in speech recognition, starting in the mid-1970s.

Later applied in several areas, including NLP and analysis of biological sequences.

HMM is a **generative model**: it models how a class could generate some input data. You might use the model to generate examples.

Contrast with discriminative models, discussed later, which only learn to distinguish classes, without learning much about them.

Hidden Markov model

Let $w_{1:n} = w_1, w_2, \ldots, w_n$ be an input sequence of words, and let $\mathcal{Y}(w_{1:n})$ be the set of all possible tag sequences $t_{1:n} = t_1, t_2, \ldots, t_n$ for $w_{1:n}$.

The goal of POS tagging is to choose the **most probable** tag sequence $\widehat{t}_{1:n} \in \mathcal{Y}(w_{1:n})$

$$\widehat{t}_{1:n} = \underset{t_{1:n} \in \mathcal{Y}(w_{1:n})}{\operatorname{argmax}} P(t_{1:n} \mid w_{1:n})$$

This is the typical formulation for a structured prediction problem. Note that $\mathcal{Y}(w_{1:n})$ is not the set of all strings of length n over the tagset.

Term $P(t_{1:n} \mid w_{1:n})$ is referred to as the **posterior** probability and can be **difficult** to model/compute.

Hidden Markov model

We break down the posterior probability using **Bayes rule**:

$$\begin{aligned}
\widehat{t}_{1:n} &= \operatorname*{argmax}_{t_{1:n} \in \mathcal{Y}(w_{1:n})} P(t_{1:n} \mid w_{1:n}) \\
&= \operatorname*{argmax}_{t_{1:n} \in \mathcal{Y}(w_{1:n})} \frac{P(w_{1:n} \mid t_{1:n}) \cdot P(t_{1:n})}{P(w_{1:n})} \\
&= \operatorname*{argmax}_{t_{1:n} \in \mathcal{Y}(w_{1:n})} P(w_{1:n} \mid t_{1:n}) \cdot P(t_{1:n})
\end{aligned}$$

where we have used the fact that $w_{1:n}$ is given, so $P(w_{1:n})$ is a constant.

The term $P(t_{1:n})$ is referred to as the **prior** or **marginal** probability. The term $P(w_{1:n} \mid t_{1:n})$ is the **likelihood** of the words given the tags.

HMM models $P(w_{1:n} \mid t_{1:n}) \cdot P(t_{1:n})$, which equals the **joint probability** $P(t_{1:n}, w_{1:n})$.

Hidden Markov model

HMM POS taggers make two simplifying assumptions.

The **first** is that the probability of a word depends only on its own POS tag and is independent of neighboring words and tags:

$$P(w_{1:n} \mid t_{1:n}) \approx \prod_{i=1}^{n} P(w_i \mid t_i)$$

The factor $P(w_i \mid t_i)$ is referred to as the **emission** probability.

The emission probability answers the following questions: If we were going to generate t_i, how likely is it that the associated word would be w_i?

Hidden Markov model

The **second** assumption is the Markov assumption that the probability of a tag is dependent only on the previous tag, rather than the entire tag sequence

$$P(t_{1:n}) \approx \prod_{i=1}^{n} P(t_i \mid t_{i-1})$$

This is connected with 2-gram models. The right-hand side requires start- and end-markers which are ignored here for simplicity and will be discussed later.

The factor $P(t_i \mid t_{i-1})$ is referred to as the **transition** probability.

Hidden Markov model

Putting everything together we have:

$$\begin{aligned}
\widehat{t}_{1:n} &= \operatorname*{argmax}_{t_{1:n} \in \mathcal{Y}(w_{1:n})} P(w_{1:n} \mid t_{1:n}) \cdot P(t_{1:n}) \\
&\approx \operatorname*{argmax}_{t_{1:n} \in \mathcal{Y}(w_{1:n})} \left(\prod_{i=1}^{n} P(w_i \mid t_i) \cdot P(t_i \mid t_{i-1}) \right)
\end{aligned}$$

Probability estimation

Probability estimation

Assume a **tagged corpus**, where each word has been tagged with its gold label. We implement **supervised** learning using the relative frequency estimator.

See lecture on language model for definition of count $C(\cdot)$.

For the transition probability we obtain:

$$P(t_i \mid t_{i-1}) = \frac{C(t_{i-1} t_i)}{C(t_{i-1})}$$

Similarly, for the emission probability we obtain:

$$P(w_i \mid t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

Probability estimation

Example :

$$P(\text{NN} \mid \text{DT}) = \frac{C(\text{DT NN})}{C(\text{DT})} = \frac{56509}{116454} \approx 0.49$$

$$P(\text{is} \mid \text{VBZ}) = \frac{C(\text{VBZ}, \text{is})}{C(\text{VBZ})} = \frac{10073}{21627} \approx 0.47$$

Using the WSJ tagset: VBZ = verb 3rd singular, NN = singular noun, DT = determiner.

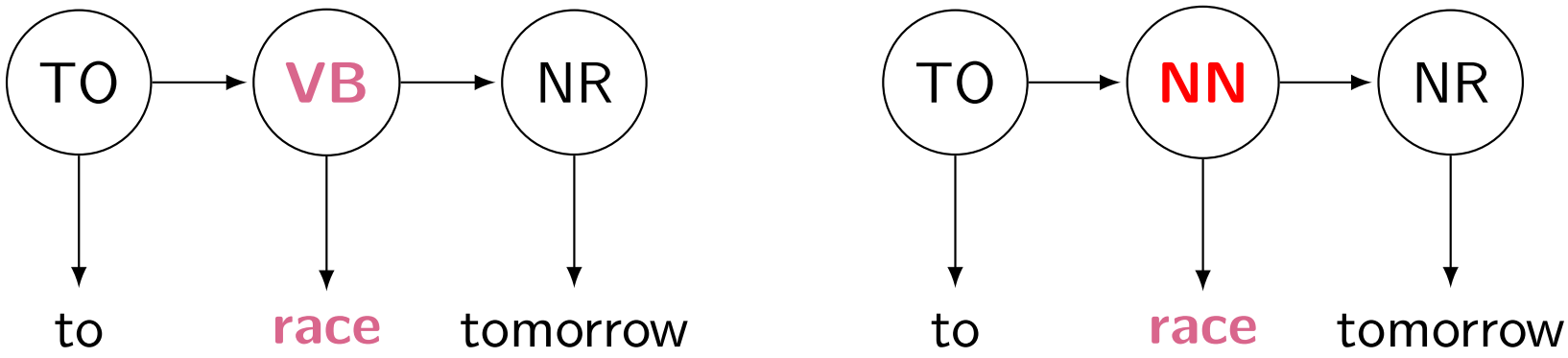
Probability estimation

Example :

He/PPS is/VBZ expected/VBN to/TO **race/VB** tomorrow/NR

Why is VB more likely than NN for token **race** in sentence above?

There are at least two sequences of tags which we could consider:



Probability estimation

The two emission probabilities turn out not to differ too much:

$$P(\textbf{race} \mid \textbf{VB}) = 0.00012$$
$$P(\textbf{race} \mid \textbf{NN}) = 0.00057$$

Nor is there much difference on whether NR follows NN or VB:

$$P(\text{NR} \mid \textbf{VB}) = 0.0027$$
$$P(\text{NR} \mid \textbf{NN}) = 0.0012$$

However, the big difference is in whether NN or VB follows TO:

$$P(\textbf{VB} \mid \text{TO}) = 0.83$$
$$P(\textbf{NN} \mid \text{TO}) = 0.00047$$

Probability estimation

Altogether we get:

$$P(\textbf{VB} \mid \text{TO})\ P(\textbf{race} \mid \textbf{VB})\ P(\text{NR} \mid \textbf{VB}) = 0.00000027$$
$$P(\textbf{NN} \mid \text{TO})\ P(\textbf{race} \mid \textbf{NN})\ P(\text{NR} \mid \textbf{NN}) = 0.00000000032$$

therefore VB is the more likely tag for **race** in this sentence, assuming the preceding one is TO and the next one is NR.

HMMs as automata

©Pexels

HMMs as automata

We can formally define an HMM as a special type of **probabilistic finite state automaton** which generates sentences (instead of accepting sentences).

The states represent 'hidden' information, that is, the POS tags which are not observed.

Special start and final states are also used which are not POS tags.

The transition function is defined according to the transition probabilities.

Each state **generates** a word according to the emission probabilities. The generated output is an observable word sequence.

HMMs as automata

HMM definition:

- finite set of **output symbols** V
- finite set of **states** Q, with initial state q_0 and final state q_f
- **transition probabilities** $a_{q,q'}$ for each pair q, q',
  $q \in Q \setminus \{q_f\}$, $q' \in Q \setminus \{q_0\}$
- **emission probabilities** $b_q(u)$ for each pair q, u,
  $q \in Q \setminus \{q_0, q_f\}$, $u \in V$

Textbook uses initial state distribution π_q, that is, we have $a_{q_0,q} = \pi_q$.

HMMs as automata

Transition and emission probabilities are **subject to**:

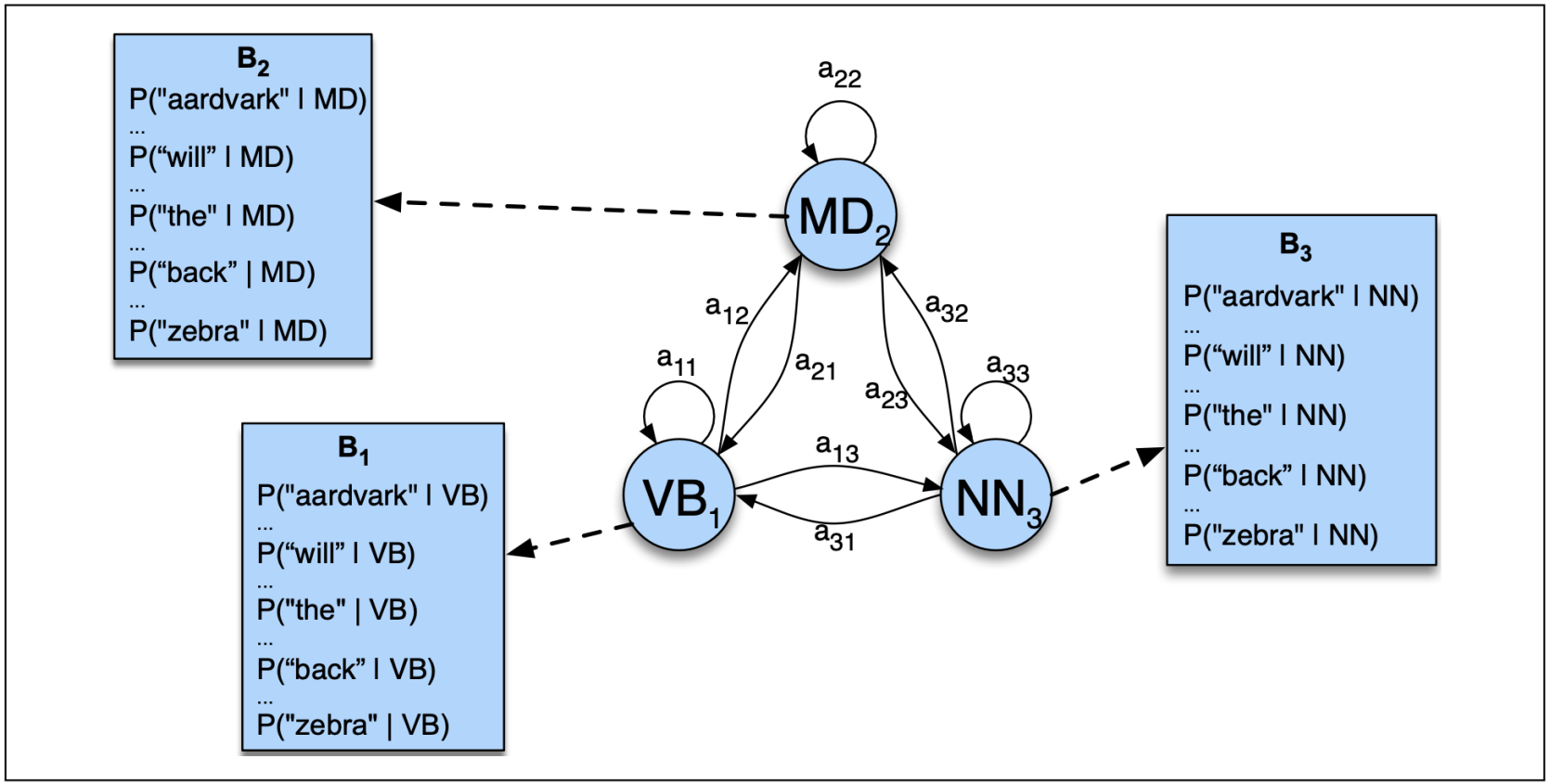
- $\sum_{q'} a_{q,q'} = 1$ for all $q \in Q \setminus \{q_f\}$
- $\sum_{u} b_q(u) = 1$ for all $q \in Q \setminus \{q_0, q_f\}$

Probabilities $a_{q_0,q}$ define the so-called **initial** probability distribution, sometimes also denoted as $\pi(q)$.

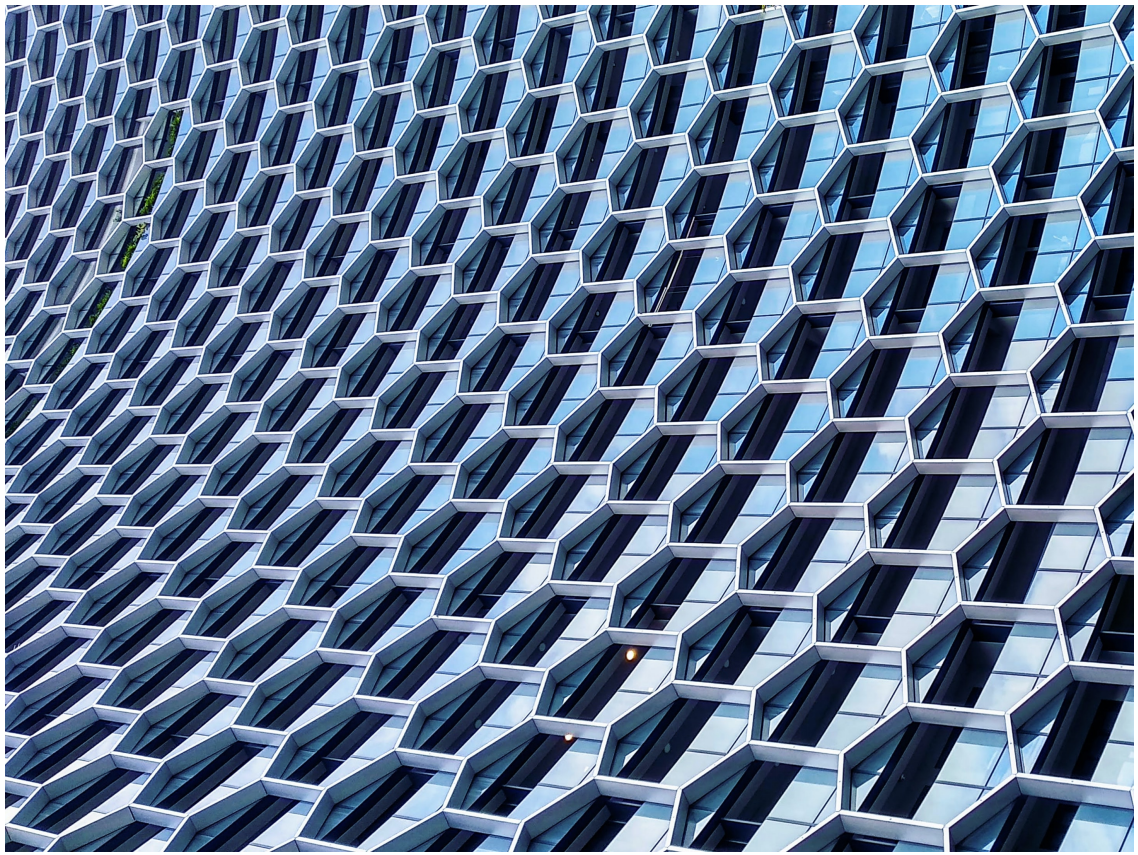
Probabilities a_{q,q_f} are the **stop** probabilities, not used in the textbook.

HMMs as automata

Example : Small excerpt, q_0 ad q_f not shown:



Viterbi algorithm



Ankit Dembla from Unsplash

Decoding

Decoding problem for HMMs: Given a sequence of observations $w_{1:n}$, find the most probable sequence of states/tags $\widehat{t}_{1:n}$

$$\begin{aligned}\widehat{t}_{1:n} &= \underset{t_{1:n} \in \mathcal{Y}(w_{1:n})}{\operatorname{argmax}} P(t_{1:n} \mid w_{1:n}) \\ &= \underset{t_{1:n} \in \mathcal{Y}(w_{1:n})}{\operatorname{argmax}} \left(\prod_{i=1}^{n} P(w_i \mid t_i) \cdot P(t_i \mid t_{i-1}) \right)\end{aligned}$$

Also called **inference** problem.

In the context of systems with hidden variables, the decoding problem asks to retrieve some hidden structure underlying the observed (input) structure.

Decoding

In order to compute $\widehat{t}_{1:n}$ we can use the following **naive algorithm**

- enumerate all sequences (paths) of POS tags $t_{1:n}$ consistent with observed sentence $w_{1:n}$
- perform a max search over all the joint probabilities $P(t_{1:n}, w_{1:n})$

This algorithm requires **exponential time**, since there can be exponentially many $t_{1:n}$ sequences in $\mathcal{Y}(w_{1:n})$!

This is a very common scenario for structured prediction problems.

Viterbi algorithm

The classical decoding algorithm for HMMs is the **Viterbi algorithm**, an instance of dynamic programming.

Related to algorithms for computing minimum edit distance.

The Viterbi algorithm computes the optimal sequence $\widehat{t}_{1:n}$ and the associated joint probability $P(\widehat{t}_{1:n}, w_{1:n})$ in **polynomial time**, exploiting dynamic programming.

Viterbi algorithm

Let $w_{1:n} = w_1, w_2, \ldots, w_n$ be the input sequence.

In what follows

- q denotes a state/tag of the HMM
- i denotes an input position, $0 \leqslant i \leqslant n + 1$

Input positions 0 and $n + 1$ represent start and end markers, respectively.

We use a two-dimensional table $vt[q, i]$ denoting the probability of the **best path** to get to state q after scanning $w_{1:i}$.

We use a two-dimensional table $bkpt[q, i]$ for retrieving the best path.

Viterbi algorithm

Initialisation step: for all q

- $vt[q, 1] = a_{q_0,q} \cdot b_q(w_1)$
- $bkpt[q, 1] = q_0$

Recursive step: for all $i = 2, \ldots, n$ and for all q

- $vt[q, i] = \max_{q'} \; vt[q', i-1] \cdot a_{q',q} \cdot b_q(w_i)$
- $bkpt[q, i] = \text{argmax}_{q'} \; vt[q', i-1] \cdot a_{q',q} \cdot b_q(w_i)$

Termination step:

- $vt[q_f, n+1] = \max_{q'} \; vt[q', n] \cdot a_{q',q_f}$
- $bkpt[q_f, n+1] = \text{argmax}_{q'} \; vt[q', n] \cdot a_{q',q_f}$

Viterbi algorithm

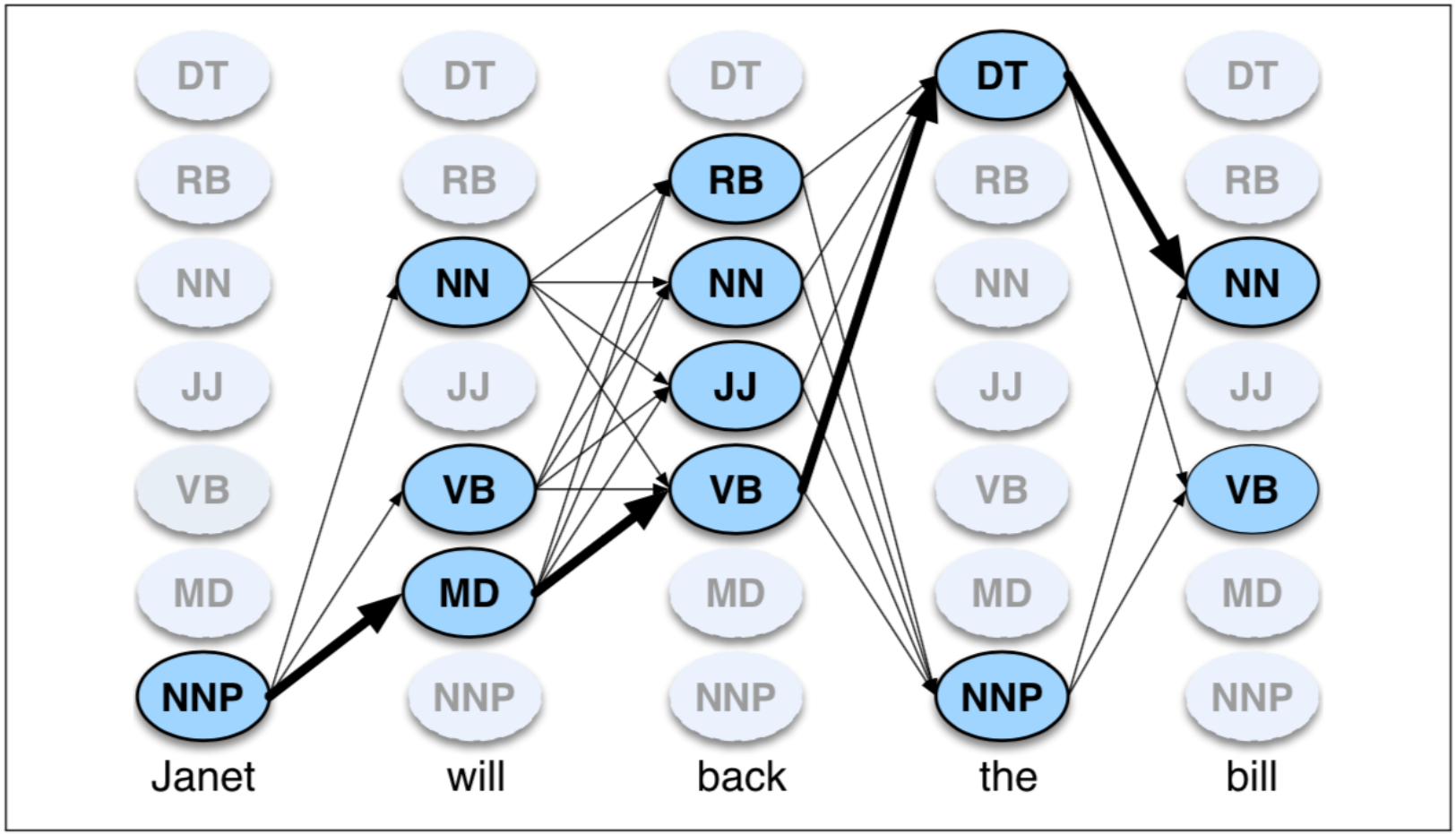
After execution of the algorithm we have

$$vt[q_f, n+1] \quad = \quad P(\hat{t}_{1:n}, w_{1:n})$$

where $\hat{t}_{1:n} = q_1, \ldots, q_n$ is the **most likely sequence** of tags for $w_{1:n}$.

The sequence of tags $\hat{t}_{1:n}$ can be **reconstructed** starting with $bkpt[q_f, n+1]$ and following the backpointers.

Example



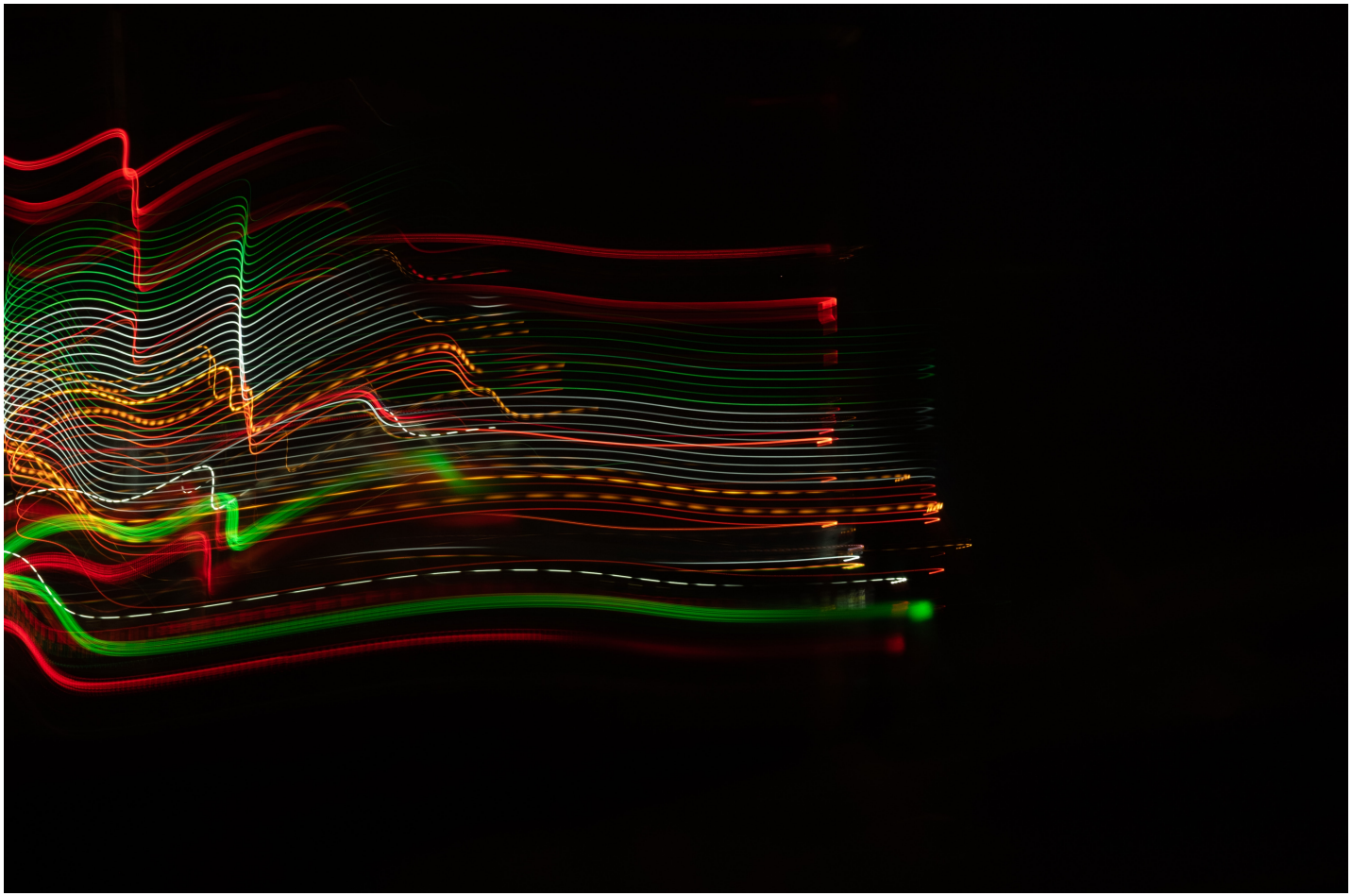
The above graph is called trellis, we will come back to this structure later.

Limitations

One limitation with HMM is that the models are exclusively run **left-to-right**.

Bidirectional models are quite standard for deep learning, as we will see with the BiLSTM models to be introduced later.

Neural POS tagger



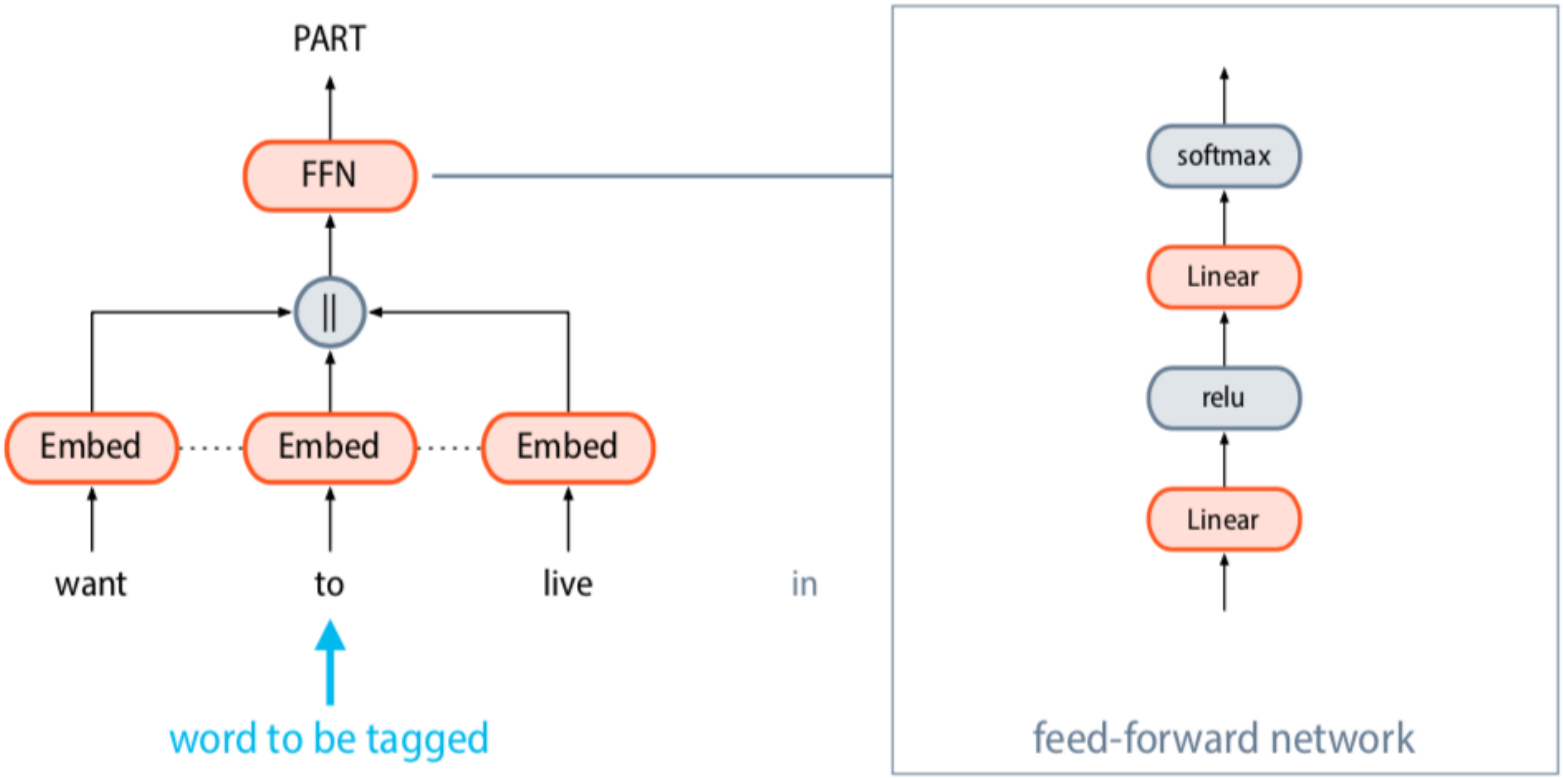
Omid Armin from Unsplash

Neural POS tagger

In neural network approaches to POS tagging, we construct distributed feature representations (dense vectors) for each tagging decision, based on the word and its context.

Neural networks can perform POS tagging as a per-token classification decision.

Fixed-window models use a feed-forward neural network to implement local search.



Second layer maps the feature space into the tagset space.

The fixed-window model is very **efficient**, since it limits the context from which information can be extracted.

Same limitation of Markov models.

Sliding windows makes it **difficult** for network to learn systematic patterns arising from phenomena like constituency, since patterns are shifted to different positions.

Let $x_1, x_2, \ldots, x_n$ be the input word sequence and $y_1, y_2, \ldots, y_n$ be the associated output POS tags.

We assume word embeddings $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$ for $x_1, x_2, \ldots, x_n$.

Recurrent neural networks can be generally described as implementing the following recursive relation, for $t = 1, \ldots, n$:

$$\begin{aligned} \mathbf{h}_t &= f(g(\mathbf{e}_t, \mathbf{h}_{t-1})) \\ &= f(\mathbf{W}^h \mathbf{h}_{t-1} + \mathbf{W}^e \mathbf{e}_t + \mathbf{b}) \end{aligned}$$

with $\mathbf{W}^h$, $\mathbf{W}^e$, $\mathbf{b}$ learnable parameters and f some nonlinear component.

Gated RNNs such as LSTM networks implement more complex recurrence relations.

We score each POS tag y by means of a **linear scalar** function of hidden state vector $\mathbf{h}_t$, and then retrive the highest score tag for x_t:

$$\begin{aligned} \psi(y, \mathbf{h}_t) &= \boldsymbol{\beta}_y \cdot \mathbf{h}_t \\ \widehat{y}_t &= \operatorname*{argmax}_y \ \psi(y, \mathbf{h}_t) \end{aligned}$$

The score $\psi(y, \mathbf{h}_t)$ can also be converted into a probability using the softmax operation:

$$P(y \mid x_{1:t}) = \frac{\exp \psi(y, \mathbf{h}_t)}{\sum_{y'} \exp \psi(y', \mathbf{h}_t)}$$

Cross-entropy or hinge loss can be used as objective function.

Recurrent bidirectional model

Hidden state vector $\mathbf{h}_t$ encodes left context up to position t but it **ignores** subsequent tokens, which might be relevant to y_t as well.

Bidirectional RNN are used to address this problem:

$$\begin{aligned} \overrightarrow{\mathbf{h}}_t &= g(\mathbf{e}_t, \overrightarrow{\mathbf{h}}_{t-1}) \\ \overleftarrow{\mathbf{h}}_t &= g'(\mathbf{e}_t, \overleftarrow{\mathbf{h}}_{t+1}) \\ \mathbf{h}_t &= [\overrightarrow{\mathbf{h}}_t; \overleftarrow{\mathbf{h}}_t] \end{aligned}$$

The scoring function $\psi(y, \mathbf{h}_t)$ is then applied to the concatenation of the two vectors.

The model is still based on **local search**, each tagging decision is made independently and no global search is performed.

Morphologically rich languages

Morphologically rich languages (MRL) have much more information than English coded into word morphology, like **case** (nominative, accusative, genitive) or **gender** (masculine, feminine).

This information is really important for tasks like parsing and coreference resolution.

Tagsets for MRL are therefore sequences of morphological features rather than a single primitive tag.

Instances of MRL are Arabic, Czech, Hungarian, Turkish, etc. Tagsets for these languages can be 4 to 10 times larger than English.

With such large tagsets, **specialized** POS taggers need to be developed where each word needs to be morphologically analyzed.

Sequence labelling

Matt Briney on Unsplash

Sequence labelling

POS tagging is an instance of a more general problem called **sequence labelling**, assigning to an input word sequence $x_1, x_2, \ldots, x_n$ an output sequence $y_1, y_2, \ldots, y_n$ over an arbitrary set of categories.

Again, this is a structured prediction problem, and categories must be assigned contextually.

We are going to briefly overview other NLP tasks that can be cast as sequence labelling problems.

Named entity recognition

Named entity recognition (NER) seeks to locate multi-word expressions referring to entities such as person names, organizations, locations, time expressions, quantities, monetary values, etc.

NER is a useful first step in lots of natural language understanding tasks.

Example : [$_{\text{PER}}$ Jane Villanueva] of [$_{\text{ORG}}$ United], a unit of [$_{\text{ORG}}$ United Airlines Holding], said the fare applies to the [$_{\text{LOC}}$ Chicago] route.

Besides tagging, in NER we also need to find the proper **span** of the target expression.

Named entity recognition

The standard approach for span-recognition is BIO tagging.

In **BIO tagging**

- tokens that begin a span are marked with label B
- tokens that occur inside a span in a position other than the leftmost one are marked with label I
- tokens outside of any span of interest are marked with O

Spans never overlap.

Example

BIO tagging along with alternative tagging techniques for span.

Words	IO Label	BIO Label	BIOES Label
Jane	I-PER	B-PER	B-PER
Villanueva	I-PER	I-PER	E-PER
of	O	O	O
United	I-ORG	B-ORG	B-ORG
Airlines	I-ORG	I-ORG	I-ORG
Holding	I-ORG	I-ORG	E-ORG
discussed	O	O	O
the	O	O	O
Chicago	I-LOC	B-LOC	S-LOC
route	O	O	O
.	O	O	O

Note the difference between the sequences BI and BB, indicating one 2-word expression vs. two 1-word expressions.

Datasets

CoNLL-2003

Named entity recognition dataset released as a part of CoNLL-2003 shared task: language-independent named entity recognition. Covering two languages: English and German.

WikiNER Dataset

Manually-labelled Wikipedia articles across nine languages: English, German, French, Polish, Italian, Spanish, Dutch, Portuguese and Russian.

Many more datasets in kaggle: `https://www.kaggle.com`.

Evaluation

Evaluation

Named entity recognizers are evaluated by precision, recall, and F1-score.

Precision is the percentage of named entities found by the learning system that are correct.

Recall is the percentage of named entities present in the corpus that are found by the system.

F1-score is the harmonic mean of the two.

Evaluation

More **specialized evaluation** can be obtained by considering individual tokens rather than entire entities.

Precision, recall and F1-score are computed for each individual BIO label. This accounts for

- assignment of wrong entity types
- wrong entity boundaries

In this **multi-class** setting, people also distinguish between different types of per-class F1-score

- macro averaged (unweighted mean)
- micro averaged (accuracy)
- weight averaged (considering each class support)

Aspect-based sentiment analysis

Aspect-based sentiment analysis aims to identify the aspects of the entities being reviewed and to determine the sentiment the reviewers express for each aspect.

More fine-grained task than sentiment analysis.

Word segmentation

Word segmentation is the task of dividing a text into its component words.

Languages which do not have a trivial word segmentation process include Chinese and Japanese, where words are not delimited.

我	有	一 台	计 算 机	。
I	have	a	computer	.

Text anonymization

Text anonymization masks sensible information from text.

This is often done to comply with the General Data Protection Regulation (GDPR) and other relevant legislation around the world.

PERSON_FIRSTNAME_2 PERSON_LASTNAME_1 is an LOCATION_1 singer, songwriter, actor and film producer who was born on DATE_1 DATE_1 DATE_1 and is now NUMERIC_1 years old. PERSON_FIRSTNAME_2 PERSON_LASTNAME_1 is the lead singer of rock band, ORGANIZATION_2. PERSON_FIRSTNAME_2 PERSON_LASTNAME_1 is known as a rock legend and for PRONOUN charismatic stage presence and dancing. So much so, that ORGANIZATION_1 released a song after PRONOUN dancing, called OTHER_IDENTIFYING_ATTRIBUTE_1 like PERSON_LASTNAME_1'. PERSON_FIRSTNAME_2 PERSON_LASTNAME_3 has NUMERIC_4 children, and has had multiple partners, and NUMERIC_1 spouse. PERSON_FIRSTNAME_2 PERSON_LASTNAME_1 has been with PRONOUN current partner PERSON_FIRSTNAME_1 PERSON_LASTNAME_2 since DATE_2.

Code switching

Code switching is the phenomenon of switching between languages in speech and in text.

Quite common in online social media.

Code switching can be viewed as a sequence labelling problem, where the goal is to label tokens representing **switch points**.

Rule-based systems

While machine learning sequence models are the norm in academic research, many commercial approaches are often based on **hand-written** lists of rules, with some smaller amount of supervised machine learning.

This is especially true for NER.

One common approach is to

- make repeated rule-based passes over a text, starting with rules with very high precision but low recall
- use machine learning methods in subsequent stages, that take the output of the first pass into account