Natural Language Processing: Part II (Paper 10)
Overview of Natural Language Processing (L90): Part III/ACS

2019, 12 Lectures, Michaelmas Term (these notes, lectures 1 to 4)
September 26, 2019
Lectured by Ryan Cotterell and Paula Buttery

Lecture Synopsis

Aims This course introduces the fundamental techniques of natural language processing. It aims to explain the potential and the main limitations of these techniques. Some current research issues are introduced and some current and potential applications discussed and evaluated.

1. **Introduction.** Brief history of NLP research, some current applications, components of NLP systems.
2. **Finite-state techniques.** Inflectional and derivational morphology, finite-state automata in NLP, finite-state transducers.
3. **Prediction and part-of-speech tagging.** Corpora, simple N-grams, word prediction, stochastic tagging, evaluating system performance.
4. **Context-free grammars and parsing.** Generative grammar, context-free grammars, parsing with context-free grammars, weights and probabilities. Some limitations of context-free grammars.
5. **Dependencies.** Dependency structures. English as an outlier. Universal dependencies. Introduction to dependency parsing.
6. **Compositional semantics.** Compositional semantics. Logical representations. Compositional semantics and lambda calculus. Inference and robust entailment. Negation.
7. **Lexical semantics.** Semantic relations, WordNet, word senses. Grounding.
8. **Distributional semantics** Representing lexical meaning with distributions. Similarity metrics.
9. **Distributional semantics and deep learning.** Embeddings. Multimodal systems and visual question answering.
10. **Discourse and dialogue.** Anaphora resolution, summarization.
11. **Language generation and regeneration.** Components of a generation system. Summarisation. Generation of referring expressions.
12. **Recent NLP research.** Some recent NLP research.

Objectives

At the end of the course students should

- be able to discuss the current and likely future performance of several NLP applications;
- be able to describe briefly a fundamental technique for processing language for several subtasks, such as morphological processing, parsing, summarization etc;
- understand how these techniques draw on and relate to other areas of computer science.

Overview

NLP is a large and multidisciplinary field, so this course can only provide a very general introduction. The idea is that this is a 'taster' course that gives an idea of the different subfields and shows a few of the huge range of computational techniques that are used. The first lecture is designed to give an overview including a very brief idea of the main applications and the methodologies which have been employed. The history of NLP is briefly discussed as a way of putting this into perspective. The next nine lectures describe some of the main subdisciplines in more detail. The organisation is roughly based on increased 'depth' of processing, starting with relatively surface-oriented techniques and progressing to considering meaning of sentences and meaning of utterances in context. Most lectures will start off by considering the subarea as a whole and then go on to describe one or more sample algorithms which tackle particular problems. The algorithms have been chosen because they are relatively straightforward to describe and because they illustrate a specific technique which has been shown to be useful, but the idea is to exemplify an approach, not to give a detailed survey (which would be impossible in the time available). Lectures 2-10 are primarily (though not exclusively) about analysing language: lecture 11 discusses generation. The final lecture will be about some current NLP research. The material in Lecture 12 will not be directly examined. It is unlikely there will be any notes for Lecture 12, but slides will be made available via the course webpage after the lecture.

There are various themes running throughout the lectures. One theme is the connection to linguistics and the tension that sometimes exists between the predominant view in theoretical linguistics and the approaches adopted within NLP. A somewhat related theme is the distinction between knowledge-based and probabilistic approaches. Evaluation will be discussed in the context of the different algorithms.

Because NLP is such a large area, there are many topics that aren't touched on at all in these lectures. Speech recognition and speech synthesis is almost totally ignored. Information Retrieval is the topic of a separate Part II course.

Feedback on the handout, lists of typos etc, would be greatly appreciated.

Recommended Reading

Recommended Book:

Jurafsky, Daniel and James Martin, *Speech and Language Processing*, Prentice-Hall, 2008 (second edition): referenced as J&M throughout this handout. This doesn't have anything on language generation (lecture 10): there is a chapter on this in the first edition of J&M but it is not useful for this course.

Background:

These books are about linguistics rather that NLP/computational linguistics. They are not necessary to understand the course, but should give readers an idea about some of the properties of human languages that make NLP interesting and challenging, without being technical.

Pinker, S., *The Language Instinct*, Penguin, 1994.

This is a thought-provoking and sometimes controversial 'popular' introduction to linguistics.

Matthews, Peter, *Linguistics: a very short introduction*, OUP, 2003.

The title is accurate ...

Background/reference:

The Internet Grammar of English, `http://www.ucl.ac.uk/internet-grammar/home.htm`

Syntactic concepts and terminology.

Bender, Emily M. 2013. *Linguistic Fundamentals for Natural Language Processing: 100 Essentials from Morphology and Syntax.* Synthesis Lectures on Human Language Technologies #20. Morgan and Claypool Publishers.

This is based on a tutorial which was entitled '100 things you always wanted to know about Linguistics but were afraid to ask'[1]. It is an extremely succinct introduction to the concepts in morphology and syntax most relevant to computational linguistics, looking at a wide range of languages.

[1] '... for fear of being told 1000 more'

Study Guide

These lectures are available to ACS/Part III students (as part of module L90, 'Overview of Natural Language Processing') as well as to Part II students, as part of Paper 10. The other component of both courses are two different practical exercises (one for ACS/Part 3 (L90) and the other for Part 2/Paper 10), which will be described in separate documents.

The handout is not intended to substitute for a textbook (or for thought). In most cases, J&M go into a considerable amount of further detail: rather than put lots of suggestions for further reading in the handout, in general the assumption here was that students will look at J&M, and then follow up the references in there if they are interested. The notes at the end of each lecture give details of the sections of J&M that are relevant and details of any discrepancies with these notes.

Generally this course is taking a rather informal/example-based approach to concepts such as finite-state automata, context-free grammars etc. The assumption is that students will have already covered this material in other contexts and that this course will illustrate some NLP applications.

This course inevitably assumes some very basic linguistic knowledge, such as the distinction between the major parts of speech. It introduces some linguistic concepts that won't be familiar to all students: since the lectures have to go through these quickly, reading the first few chapters of an introductory linguistics textbook may help students understand the material. The idea is to introduce just enough linguistics to motivate the approaches used within NLP rather than to teach the linguistics for its own sake. At the end of this handout, there are some mini-exercises to help students understand the concepts: it would be very useful if these were attempted before the lectures as indicated. There are also some suggested post-lecture exercises: answers to these should be discussed with the lecturer by email.

Of course, lecturers will be happy to try and answer questions about the course or more general NLP questions, preferably by email.

URLs

Nearly all the URLs given in these notes should be linked from:
`http://www.cl.cam.ac.uk/~aac10/stuff.html`
(apart from this one of course ...). If any links break, let one of the lecturers know.

1 Lecture 1: Introduction to NLP

The primary aim of this lecture is to give students some idea of the objectives of NLP. The main subareas of NLP will be introduced, especially those which will be discussed in more detail in the rest of the course. There will be a preliminary discussion of the main problems involved in language processing by means of examples taken from NLP applications. This lecture also introduces some methodological distinctions and puts the applications and methodology into some historical context.

1.1 What is NLP?

Natural language processing (NLP) can be defined as the computational modelling of human language. The term 'NLP' is sometimes used rather more narrowly than that, often excluding information retrieval and sometimes even excluding machine translation. NLP is sometimes contrasted with 'computational linguistics' (CL), with NLP being thought of as more applied. Alternative terms are often preferred, like 'Language Technology' or 'Language Engineering'. The term 'language' is often used in contrast with 'speech' (e.g., Speech and Language Technology). But I'm going to simply refer to NLP and use the term broadly.

NLP is essentially multidisciplinary: it is closely related to linguistics (although the extent to which NLP overtly draws on linguistic theory varies considerably). Like NLP, formal linguistics deals with the development of models of human languages, but some approaches in linguistics reject the validity of statistical techniques, which are seen as an essential part of computational linguistics. NLP also has links to research in cognitive science, psychology, philosophy and maths (especially logic). Within CS, it relates to machine learning, formal language theory, compiler techniques, theorem proving and human-computer interaction. Of course it is also related to AI. The great majority of NLP research involves machine learning techniques to some extent. The NLP research community has its own conferences (lots of them ...) and journals but some NLP research is published in general AI or Machine Learning conferences and the relationship is probably becoming closer.

1.2 Some linguistic terminology

The course is organised so that there are ten lectures after this one corresponding to different NLP subareas. In the first eight, the discussion moves from relatively 'shallow' processing to areas which involve meaning and connections with the real world. These subareas loosely correspond to some of the standard subdivisions of linguistics:

1. Morphology: the structure of words. For instance, *unusually* can be thought of as composed of a prefix *un-*, a stem *usual*, and an affix *-ly*. *composed* is *compose* plus the inflectional affix *-ed*: a spelling rule means we end up with *composed* rather than *composeed*. Morphology will be discussed in lecture 2.
2. Syntax: the way words are used to form phrases. e.g., it is part of English syntax that a determiner (a word such as *the*) will come before a noun, and also that determiners are obligatory with certain singular nouns. Formal and computational aspects of syntax will be discussed in lectures 3, 4 and 5.
3. Semantics. Compositional semantics is the construction of meaning (often expressed as logic) based on syntax. This is discussed in lecture 6. This is contrasted to lexical semantics, i.e., the meaning of individual words which is the topic of lectures 7, 8 and 9.
4. Pragmatics: meaning in context. This will come into lectures 9 and 10, although linguistics and NLP generally have very different perspectives here.

Lecture 11 looks at language generation rather than language analysis.

1.3 Why is computational language processing difficult?

Consider trying to build a system that would answer email sent by customers to a retailer selling laptops and accessories via the Internet.[2] This might be expected to handle queries such as the following:

[2] I originally wrote this section in 2003 inspired by the experience of colleagues who had built a commercial customizable email response system. The examples now look rather retro, but we still don't have much in the way of automatic email response and the general discussion still holds.

- Has my order number 4291 been shipped yet?
- Is FD5 compatible with a 505G?
- What is the speed of the 505G?

Assume the query is to be evaluated against a database containing product and order information, with relations such as the following:

ORDER

Order number	Date ordered	Date shipped
4290	2/2/13	2/2/13
4291	2/2/13	2/2/13
4292	2/2/13	

USER: Has my order number 4291 been shipped yet?
DB QUERY: order(number=4291,date_shipped=?)
RESPONSE TO USER: Order number 4291 was shipped on 2/2/13

It might look quite easy to write patterns in order to build a system to respond to these queries, but very similar strings can mean very different things, while very different strings can mean much the same thing. 1 and 2 below look very similar but mean something completely different, while 2 and 3 look very different but essentially mean the same in this context.

1. How fast is the TZ?
2. How fast will my TZ arrive?
3. Please tell me when I can expect the TZ I ordered.

While some tasks in NLP can be done adequately without having any sort of account of meaning, others require that we can construct detailed representations which will reflect the underlying meaning rather than the superficial string.

In fact, in natural languages (as opposed to programming languages), ambiguity is ubiquitous, so exactly the same string might mean different things. For instance in the query:

> Do you sell Sony laptops and disk drives?

the user may or may not be asking about Sony disk drives. This particular ambiguity may be represented by different bracketings:

> Do you sell (Sony laptops) and (disk drives)?
> Do you sell (Sony (laptops and disk drives))?

We'll see lots of examples of different types of ambiguity in these lectures.

Natural language has properties which are essential to communication which are not found in formal languages, such as predicate calculus, computer programming languages, semantic web languages and so on. Natural language is incredibly flexible. It is learnable, but compact. Natural languages are emergent, evolving systems. Ambiguity and synonymy are inherent to flexibility and learnability. Despite ambiguity, natural language can be indefinitely precise: ambiguity is largely local[3] (at least for humans) and natural languages accommodate (semi-)formal additions.

Often humans have knowledge of the world which resolves a possible ambiguity, probably without the speaker or hearer even being aware that there is a potential ambiguity.[4] But hand-coding such knowledge in NLP applications has turned out to be impossibly hard to do for more than very limited domains: the term *AI-complete* is sometimes

[3] i.e., immediate context resolves the ambiguity: examples of this will be discussed in later lectures.

[4] I'll use *hearer* generally to mean the person who is on the receiving end, regardless of the modality of the language transmission: i.e., regardless of whether it's spoken, signed or written. Similarly, I'll use *speaker* for the person generating the speech, text etc and *utterance* to mean the speech or text itself. This is the standard linguistic terminology, which recognises that spoken language is primary and text is a later development.

used (by analogy to NP-complete), meaning that we'd have to solve the entire problem of representing the world and acquiring world knowledge.[5] The term AI-complete is intended jokingly, but conveys what's probably the most important guiding principle in current NLP: we're looking for applications which don't require AI-complete solutions: i.e., ones where we can either work with very limited domains or approximate full world knowledge and reasoning by relatively simple techniques. Machine learning hasn't (yet?) got us out of this situation, for a variety of reasons, most importantly because of the need for training data.

1.4 Some NLP applications

Useful NLP systems have been built for a large range of applications, including (but not limited to):

- spelling and grammar checking
- predictive text
- optical character recognition (OCR)
- screen readers for blind and partially sighted users
- augmentative and alternative communication (i.e., systems to aid people who have difficulty communicating because of disability)
- machine aided translation (i.e., systems which help a human translator, e.g., by storing translations of phrases and providing online dictionaries integrated with word processors, etc)
- lexicographers' tools
- information retrieval
- document classification (filtering, routing)
- document clustering
- information extraction
- sentiment classification
- text mining
- question answering
- summarization
- text segmentation
- exam marking
- language teaching and assessment
- report generation (possibly multilingual)
- machine translation
- natural language interfaces to databases
- email response
- dialogue systems

[5] In this course, I will use *domain* to mean some circumscribed body of knowledge: for instance, information about laptop orders constitutes a limited domain.

Several of these applications are discussed briefly below. Roughly speaking, the list is ordered according to the complexity of the language technology required. The applications towards the top of the list can be seen simply as aids to human users, while those at the bottom may be perceived as agents in their own right. Perfect performance on any of these applications would be AI-complete, but perfection isn't necessary for utility: in many cases, useful versions of these applications had been built by the late 70s. Commercial success has often been harder to achieve, however.

1.5 An example application: Sentiment classification

Finding out what people think about politicians, policies, products, services, companies and so on is a huge business. Increasingly this is done by automatic analysis of web documents and social media. The full problem involves finding all the references to an entity from some document set (e.g., references to Hillary Clinton in all Daily Mail articles appearing in September 2014, references to Siri in all tweets with hashtag #apple), and then classifying the references as positive, negative or neutral. Customers who use opinion mining want to see summaries of the data (e.g., to see whether popularity is going up or down), but may also want to see actual examples (as text snippets). Companies generally want a fine-grained classification of aspects of their product (e.g., laptop batteries, phone screens).

A full opinion mining system requires that relevant text is retrieved, references to the objects of interest are recognized in that text (generally as *named entities*: e.g., *Sony 505G*, *Hillary Clinton*), and the parts of the text that refer to those entities are determined. Once this is done, the referring text can be classified for positive or negative sentiment. To be commercially useful, this has to be done very rapidly, especially when analyzing trends on social media, so a significant software engineering effort is involved. But academic researchers have looked at a simpler version of the task by starting from a fixed set of documents which are already known to be opinions about a particular topic or entity (e.g., reviews), where the problem is just to work out whether the author is expressing positive or negative opinions. This allows researchers to focus on sentiment classification but has still been a challenging problem to address. Some of the early research work was done on movie reviews (Pang et al, 2002).[6] The rating associated with each review is known (that is, reviewers give each movie a numerical score), so there is an objective standard as to whether the review is positive or negative. This avoids the need to manually annotate the data before experimenting with it, which is a time-consuming and error-prone process. The research problem is to assign sentiment automatically to each document in the entire corpus to agree with the known ratings.[7] Pang et al only selected review articles which had clear positive or negative ratings, and balanced their corpus so the positive/negative split was 50/50.

The most basic technique is to look at the words in the review in isolation of each other, and to classify the document on the basis of whether those words generally indicate positive or negative reviews. This is a *bag-of-words* technique: we model the document as an unordered collection of words (*bag* rather than set because there will be repetition). A document with more positive words than negative ones should be a positive review. In principle, this could be done by using human judgements of positive/negative words, but using machine learning techniques works better[8] (humans don't consider many words that turn out to be useful indicators). However, Pang et al found that the accuracy of the classification is only around 80% (for a problem where there was a 50% chance success rate). One source of errors is negation: (e.g., *Ridley Scott has never directed a bad film* is a positive statement). Another problem is that the machine learning technique may match the data too closely: e.g., if a machine learner were trained on reviews which include a lot of films from before 2005, it would discover that *Ridley* was a strong positive indicator but it would then tend to misclassify reviews for 'Kingdom of Heaven' (which was panned). More subtle problems arise from not tracking the contrasts in the discourse. The two extracts below are from Pang et al's paper:

> This film should be brilliant. It sounds like a great plot, the actors are first grade, and the supporting cast is good as well, and Stallone is attempting to deliver a good performance. However, it can't hold up.. . . [taken from a review by David Wilcock of 'Cop Land' `http://www.imdb.com/reviews/101/10185.html`]

> AN AMERICAN WEREWOLF IN PARIS is a failed attempt . . . Julie Delpy is far too good for this movie. She imbues Serafine with spirit, spunk, and humanity. This isn't necessarily a good thing, since it prevents

[6] Pang, Lee and Vaithyanatha (2002), *Thumbs up? Sentiment Classification using Machine Learning Techniques* In Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP).

[7] A *corpus* (plural *corpora*) is the technical term for a body of text that has been collected for some purpose, see §3.1.

[8] Classifiers are discussed in more detail in lecture 10.

> us from relaxing and enjoying AN AMERICAN WEREWOLF IN PARIS as a completely mindless, campy entertainment experience. Delpy's injection of class into an otherwise classless production raises the specter of what this film could have been with a better script and a better cast ... She was radiant, charismatic, and effective ... [taken from a review by James Berardinelli `http://www.imdb.com/reviews/103/10363.html`]

Unfortunately, although in principle NLP techniques can deal with syntax, semantics and discourse and thus address these sort of problems, doing this in a way that can significantly improve performance over the simple system turns out to be hard. Many 'obvious' techniques for improving the bag-of-words model did not improve performance. For instance, although a good model should include negation, it only applies to a small percentage of examples. Modelling negation requires parsing to determine the scope of negation correctly, but it is easy to introduce errors which make performance worse overall. Nevertheless, systems have been developed that have considerably better performance than the simple methods: for instance, Moilanen and Pulman (2007) [9] demonstrated that parsing and a form of compositional semantics (topics which will be introduced in Lectures 4, 5 and 6) can considerably improve performance over the simple methods. High performance has also recently been achieved by neural network methods which are believed to have induced some sort of internal representation of syntactic and/or semantic structure. In any experiment of this type, there will also be limitations due to the test data: humans do not achieve 100% performance on this task. Ultimately to understand whether a statement is really positive or negative in a particular context is AI-complete in terms of world knowledge and additionally requires modelling the state of mind of the speaker (think about irony, for instance). The (application-oriented) question is not whether automatic methods can be perfect but whether they can be good enough to be useful.

1.6 Information retrieval, information extraction and question answering

Information retrieval (IR) involves returning a set of documents in response to a user query: Internet search engines were developed from traditional IR techniques, but use additional methods such as ranking documents according to links (e.g., Google's PageRank).

Information extraction involves trying to discover specific information from a set of documents. The information required can be described as a template. For instance, for company joint ventures, the template might have slots for the companies, the dates, the products, the amount of money involved. The slot fillers are generally strings.

Question answering attempts to find a specific answer to a specific question from a set of documents, or at least a short piece of text that contains the answer.

(1) Q: What is the capital of France?
A: Paris has been the French capital for many centuries.

There have been question-answering systems on the Web since the 1990s, but until recently most used very basic techniques. One common approach involved employing a large staff of people who searched the web to find pages which are answers to potential questions. The question-answering system then performed very limited manipulation on the actual user input to map to a known question. The same basic technique is used in many online help systems. However, with enough resource, impressive results from automatic QA are now possible: most famously an IBM research team created a QA system that beat human champions on the quiz show *Jeopardy!* in 2011 (see Ferrucci et al, AI Magazine, 2010, for an overview). Current QA systems are very successful at answering questions that humans find difficult, but very bad at answering many questions that humans find trivial, essentially because such information is not explicitly stated anywhere online. The problem is that the trivial information is essential for modelling human reasoning and thus for really understanding language.

1.7 Machine translation

MT work started in the US in the early fifties, concentrating on Russian to English. A prototype system was publicly demonstrated in 1954 (remember that the first electronic computer had only been built a few years before that). MT

[9] Moilanen, Karo, and Stephen Pulman. "Sentiment composition." In Proceedings of the Recent Advances in Natural Language Processing International Conference, pp. 378-382. 2007.

funding was drastically cut in the US in the mid-60s and ceased to be academically respectable in some places, but Systran was providing useful translations by the late 60s using hand-built *transfer rules* to map between source and target languages. Systran is still going — updating the rule-based system over the years was an amazing feat of software engineering. Since 2010, they have sold a hybrid rule-based / *Statistical Machine Translation* (SMT) system. Systran was used for most of the language pairs available from Google Translate until about 2007/2008, but Google then introduced an SMT system which was developed in-house, exploiting Google's access to the huge amount of *parallel text* available on the web (i.e., they mine web pages for pairings of source text and translated text). This works very well for some language pairs and very badly for others, depending firstly on the availability of properly translated parallel text and secondly on the closeness of the languages. In fact, Google also used manually engineered rules to augment its SMT system. Recently, with the general acknowledgement that SMT techniques were reaching a plateau, with more-and-more data was needed for small improvements in performance, interest has switched to Neural Machine Translation (NMT). This allows far greater possibilities for integration of information from monolingual sources than the earlier SMT approaches. (But for some language pairs, MT systems using manually-constructed rules still greatly outperform SMT or NMT.)

Until the 80s, the utility of general purpose MT systems was severely limited by the fact that text was not available in electronic form: for instance, in the 1960s and 70s, teams of skilled typists were needed to input Russian documents before the MT system was run.

None of these systems are a substitute for good human translation, but they are now better than bad human translation. They are useful because they allow users to get an idea of what a document is about. In contexts where an SMT/NMT system can be trained on good parallel text and the human user has good contextual/domain knowledge, they are very helpful.

1.8 Natural language interfaces and dialogue systems

Natural language interfaces were the 'standard' NLP application in the 70s and 80s. LUNAR is the classic example of a natural language interface to a database (NLID): it concerned lunar rock samples brought back from the Apollo missions. LUNAR is described by Woods (1978) (but the work was done several years earlier): it was capable of translating elaborate natural language expressions into database queries.

SHRDLU (Winograd, 1973) was a system capable of participating in a dialogue about a microworld (the blocks world) and manipulating this world according to commands issued in English by the user. SHRDLU had a big impact on the perception of NLP at the time since it seemed to show that computers could actually 'understand' language: the impossibility of scaling up from the microworld was not realised.

LUNAR and SHRDLU both exploited the limitations of one particular domain to make the natural language understanding problem tractable, particularly with respect to ambiguity. To take a trivial example, if you know your database is about lunar rock, you don't need to consider the music or movement senses of *rock* when you're analysing a query.

There have been many advances in NLP since these systems were built: natural language interface systems have become much easier to build, and somewhat easier to use, but they still haven't become ubiquitous. Natural Language interfaces to databases were commercially available in the late 1970s, but largely died out by the 1990s: porting to new databases and especially to new domains requires very specialist skills and is essentially too expensive (automatic porting was attempted but never successfully developed). Users generally preferred graphical interfaces when these became available.

Modern 'intelligent' personal assistants, such as Siri, use interfaces to databases as part of their functionality, along with search and question answering. These systems have some model of dialogue context and can adapt to individual users. Just as importantly, users adapt to the systems, rephrasing queries which are not understood and reusing queries which are successful.

1.9 Some more history

Before the 1970s, most NLP researchers were concentrating on MT as an application (see above). NLP was a very early application of computer science and started about the same time as Chomsky was publishing his first major works in formal linguistics (Chomskyan linguistics quickly became dominant, especially in the US). In the 1950s and early 1960s, there was little distinction between formal linguistics and computational linguistics: ideas about formal

grammar were being worked out in linguistics, and algorithms for parsing natural language were being developed at the same time as algorithms for parsing programming languages. However, the approaches that Chomsky and his colleagues developed turned out to be only somewhat indirectly useful for NLP and the two fields became distinct, at least in the US and UK.

NLP in the 1970s and first half of the 1980s was predominantly based on a paradigm where extensive linguistic and real-world knowledge was hand-coded. There was controversy about how much linguistic knowledge was necessary for processing, with some researchers downplaying syntax, in particular, in favour of world knowledge. NLP researchers were very much part of the AI community (especially in the US and the UK), and the debate that went on in AI about the use of logic vs other meaning representations ('neat' vs 'scruffy') also affected NLP. By the 1980s, several linguistic formalisms had appeared which were fully formally grounded and reasonably computationally tractable, and the linguistic/logical paradigm in NLP was firmly established. Unfortunately, this didn't lead to many useful systems, partly because many of the difficult problems (disambiguation etc) were seen as somebody else's job (and mainstream AI was not developing adequate knowledge representation techniques) and partly because most researchers were concentrating on the 'agent-like' applications and neglecting the user aids. Although the symbolic, linguistically-based systems sometimes worked quite well as NLIDs, they proved to be of little use when it came to processing less restricted text, for applications such as IE. It also became apparent that lexical acquisition was a serious bottleneck for serious development of such systems.

Statistical NLP became the most common paradigm in the 1990s, at least in the research community. By this point, there was a huge divide between mainstream linguists and the NLP community. Chomsky had declared:

> But it must be recognized that the notion 'probability of a sentence' is an entirely useless one, under any known interpretation of this term. (Chomsky 1969)

Certain linguistics journals would not even review papers which had a quantitative component. But speech and NLP researchers wanted results:

> Whenever I fire a linguist our system performance improves. (Fred Jelinek, allegedly said at a workshop in 1988, various forms of the quotation have been attested. Jelinek later said he never actually fired anyone, they just left . . .)

Speech recognition had demonstrated that simple statistical techniques worked, given enough training data. NLP systems were built which required very limited hand-coded knowledge, apart from initial training material. Most applications were much shallower than the earlier NLIDs, but the switch to statistical NLP coincided with a change in US funding, which started to emphasise speech recognition and IE. There was also a general realization of the importance of serious evaluation and of reporting results in a way that could be reproduced by other researchers. US funding emphasised competitions with specific tasks and supplied test material, which encouraged this, although there was a downside in that some of the techniques developed were very task-specific. It should be emphasised that there had been computational work on corpora for many years (much of it by linguists): it became much easier to do corpus work by the late 1980s as disk space became cheap and machine-readable text became ubiquitous. Despite the shift in research emphasis to statistical approaches, most commercial systems remained primarily based on hand-coded linguistic information.

Later the symbolic/statistical split became less pronounced, since most researchers are interested in both.[10] Most work involves some type of machine learning, including machine learning for symbolic processing. Linguistically-based NLP made something of a comeback, with increasing availability of open source resources, and the realisation that at least some of the classic statistical techniques seem to be reaching limits on performance, especially because of difficulties of acquiring training data and in adapting to new types of text. However, modern linguistically-based NLP approaches make heavy use of machine learning and statistical processing.

Recently, the research environment has changed substantially, as a result of the use of new machine learning techniques. Deep learning[11] has radically changed speech recognition, but took somewhat longer to adopt in NLP. In some

[10] At least, there are now very few researchers who avoid statistical techniques as a matter of principle and all statistical language systems have a symbolic component!

[11] The terminology is problematic and I am using it loosely. There is a lot of disagreement about what constitutes 'deep' learning. Neural network techniques have been used for decades, but are only relatively recently more successful than alternative approaches — they have very little to do with biological neurons.

cases it has greatly improved performance compared to older techniques, but this is not true in all NLP applications. It has meant that interest in the field has massively increased, including commercial interest. The availability of toolkits has made it relatively easy to experiment. Success on vison recognition tasks has opened up a range of new experiments on grounded language and distributional semantics has been reinvented (again), allowing some knowledge from large text corpora to be effectively integrated into tasks for the first time. These developments are discussed in lecture 8. However, there will be a health warning attached: it is becoming apparent that some seemingly impressive results do not generalize as we would hope. Flaws in various datasets/tasks are becoming apparent, and one theme in this course will be the need to analyse results properly.

The ubiquity of the Internet has completely changed the space of interesting NLP applications since the early 1990s, and the vast amount of text available can potentially be exploited, especially for statistical techniques. The dotcom boom and bust at the turn of the millennium considerably affected NLP in industry. In lecture notes in 2014, I wrote that "interest has increased again" and "another boom is well under way". From the perspective of 2017, that seems like a serious understatement.

1.10 Combining components in NLP systems

Many NLP applications can be adequately implemented with relatively 'shallow' processing. For instance, English spelling checking only requires a word list and simple morphology to be useful. I'll use the term 'deep' NLP for systems that build a meaning representation (or an elaborate syntactic representation), which is generally agreed to be required for applications such as NLIDs and email question answering.[12]

The input to an NLP system could be speech or text. It could also be gesture (multimodal input or perhaps a Sign Language). The output might be non-linguistic, but most systems need to give some sort of feedback to the user, even if they are simply performing some action (issuing a ticket, paying a bill, etc). However, often the feedback can be very formulaic.

There's general agreement that certain NLP subtasks can be described semi-independently and combined to build an application, although assumptions about the detailed nature of the interfaces between the tasks differ:

- input preprocessing: speech recogniser or text preprocessor (non-trivial in languages like Chinese or for highly structured text for any language) or gesture recogniser. Such systems might themselves be very complex, but I won't discuss them in this course — we'll assume that the input to the main NLP component is segmented text.
- morphological analysis (lecture 2): this is relatively well-understood for the most common languages that NLP has considered, but is complicated for many languages (e.g., Turkish, Basque).
- part of speech tagging (lecture 3): (used by deep processing systems as a way of cutting down parser search space).
- parsing (lectures 4, 5 and 6): this includes syntax and compositional semantics, which are sometimes treated as separate subtasks.
- further semantic processing includes word sense disambiguation (lecture 7) and inference (lecture 6): these tasks are supported by lexical meaning representations (lectures 7 and 8).
- context processing: this a range of subtasks, including anaphora resolution (lecture 9).
- discourse structuring: the part of language generation that's concerned with deciding what meaning to convey (lecture 10).
- realization (lecture 10): the inverse of parsing, i.e., conversion of meaning representations or syntactic structures to strings. This may use the same grammar and lexicon[13] as the parser.
- morphological generation (lecture 2): as with morphological analysis, this is relatively straightforward for English.

[12] Here 'deep' has nothing directly to do with deep learning.

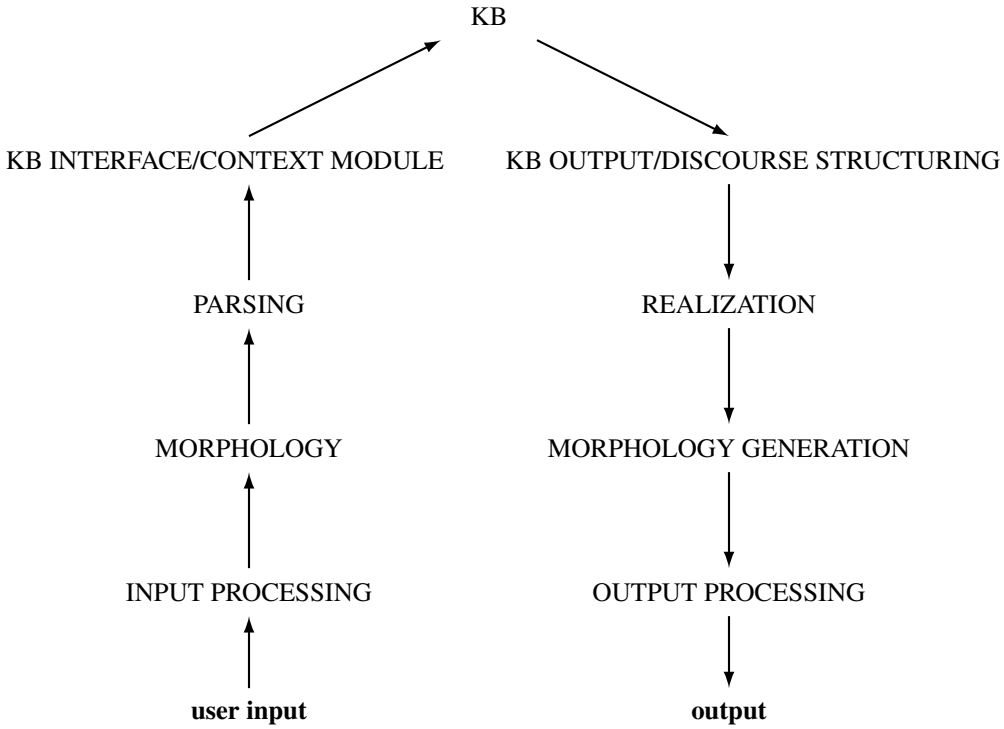
[13] The term *lexicon* is generally used for the part of the NLP system that contains dictionary-like information — i.e. information about individual words.

- output processing: text-to-speech, text formatter, etc. As with input processing, this may be simple or complex.

For a system implementing an NLP subtask to correspond to a module in an application, it obviously needs a well-defined set of interfaces. What's less obvious is that it needs its own evaluation strategy and test suites so it can be developed independently. In principle, at least, components are *reusable* in various ways: for instance, a parser could be used with multiple grammars, the same grammar can be processed by different parsers and generators, a parser/grammar combination could be used in MT or in a natural language interface. However, for a variety of reasons, it is not easy to reuse components like this, and generally a lot of work is required for each new application.

We can draw schematic diagrams for applications showing how the subtasks fit together.

Natural language interface to a knowledge base



However, this is an abstraction. Simple pipelining does not work well because each component makes errors. Much more complex architectures may be used: the IBM Jeopardy playing system has over 100 modules which produce multiple candidate answers — sophisticated probabilistic methods are used to rank these. Nevertheless, the diagram should give some indication of the potential role of each subtask in a full application.

In lectures 2–11, various algorithms will be discussed which mostly correspond to implementations of the various subtasks.

1.11 General comments

- Even 'simple' NLP applications need complex knowledge sources for some problems. Sometimes that knowledge can be partially induced from general text.
- Applications cannot be 100% perfect, because full real world knowledge and reasoning is not possible.
- Applications that are less than 100% perfect can be useful (humans aren't 100% perfect anyway).
- Applications that aid humans are much easier to construct than applications which replace humans. It may be difficult to make the limitations of 'agents' which accept speech or text input clear to naive human users.
- NLP interfaces are nearly always competing with a non-language based approach.

- Currently nearly all applications either do relatively shallow processing on arbitrary input or deep processing on narrow domains. MT can be domain-specific to varying extents: MT on arbitrary text still isn't very good (in general), but can be useful.
- Limited domain systems require extensive and expensive expertise to port or large amounts of domain-specific data. Research that relies on extensive hand-coding of knowledge for small domains is now generally regarded as a dead-end, though reusable hand-coding is a different matter.
- The development of NLP has been driven as much by hardware and software advances, and societal and infrastructure changes as by great new ideas. Improvements in NLP techniques are generally incremental rather than revolutionary. Deep learning may turn out to be an exception, but some of the hype comes from lack of awareness of earlier work or over-selling experimental results based on limited evaluations.

2 Lecture 2: Morphology and finite-state techniques

This lecture starts with a brief discussion of morphology, concentrating mainly on English morphology. The concept of a lexicon in an NLP system is discussed with respect to morphological processing. Spelling rules are introduced and the use of finite state transducers to implement spelling rules is explained. The lecture concludes with a brief overview of some other uses of finite state techniques in NLP.

2.1 A very brief and simplified introduction to morphology

Morphology concerns the structure of words. Words are assumed to be made up of *morpheme*s, which are the minimal information carrying unit. Morphemes which can only occur in conjunction with other morphemes are *affix*es: words are made up of a stem (more than one in the case of compounds) and zero or more affixes. For instance, *dog* is a stem which may occur with the plural suffix *+s* i.e., *dogs*. The compound *bookshop* has two stems (*book* and *shop*): most English compounds are spelled with a space, however, and therefore not standardly analysed by morphological processors. English only has suffixes (affixes which come after a stem) and prefixes (which come before the stem — in English prefixes are limited to derivational morphology), but other languages have *infixes* (affixes which occur inside the stem) and circumfixes (affixes which go around a stem, such as the *ge-t* in German *gekauft*). For instance, Arabic has stems (root forms) such as *k_t_b*, which are combined with infixes to form words (e.g., *kataba*, he wrote; *kotob*, books). Some English irregular verbs show a relic of inflection by infixation (e.g. *sing*, *sang*, *sung*) but this process is no longer *productive* (i.e., it won't apply to any new words, such as *ping*).[14]

Note the requirement that a morpheme can be regarded as a unit. There are cases where there seems to be a similarity in meaning between some clusters of words with similar spellings: e.g., *slink*, *slide*, *slither*, *slip*. But such examples cannot be decomposed (i.e., there is no *sl-* morpheme) because the rest of the word does not stand as a unit.

2.2 Inflectional vs derivational morphology

Inflectional and derivational morphology can be distinguished, although the dividing line isn't always sharp. The distinction is of some importance in NLP, since it means different representation techniques may be appropriate. Inflectional morphology can be thought of as setting values of slots in some *paradigm* (i.e., there is a fixed set of slots which can be thought of as being filled with simple values). Inflectional morphology concerns properties such as tense, aspect, number, person, gender, and case, although not all languages code all of these: English, for instance, has very little morphological marking of case and gender. Derivational affixes, such as *un-*, *re-*, *anti-* etc, have a broader range of semantic possibilities (there seems no principled limit on what they can mean) and don't fit into neat paradigms. Inflectional affixes may be combined (though not in English). However, there are always obvious limits to this, since once all the possible slot values are 'set', nothing else can happen. In contrast, there are no obvious limitations on the number of derivational affixes (*antidisestablishmentarianism*, *antidisestablishmentarianismization*) and they may even be applied recursively (*antiantimissile*). In some languages, such as the Inuit language(s), derivational morphology is often used where English would use adjectival modification or other syntactic means. This leads to very long 'words' occurring naturally and is responsible for the (misleading/mistaken) claim that 'Eskimo' has hundreds of words for snow.

Inflectional morphology is generally close to fully productive, in the sense that a word of a particular class will show all the possible inflections although the actual affix used may vary. For instance, an English verb will have a present tense form, a 3rd person singular present tense form, a past participle and a passive participle (the latter two being the same for regular verbs). This will also apply to any new words which enter the language: e.g., *text* as a verb — *texts*, *texted*. Derivational morphology is less productive and the classes of words to which an affix applies is less clearcut. For instance, the suffix *-ee* is relatively productive (*textee* sounds plausible, meaning the recipient of a text message, for instance), but doesn't apply to all verbs (?*snoree*, ?*jogee*, ?*dropee*). Derivational affixes may change the part of speech of a word (e.g., *-ise*/*-ize* converts nouns into verbs: *plural*, *pluralise*). However, there are also examples of what is sometimes called *zero derivation*, where a similar effect is observed without an affix: e.g. *tango*, *waltz* etc are words which are basically nouns but can be used as verbs.

[14] Arguably, though, spoken English has one productive infixation process, exemplified by *absobloodylutely*.

Stems and affixes can be individually ambiguous. There is also potential for ambiguity in how a word form is split into morphemes. For instance, *unionised* could be *union -ise -ed* or (in chemistry) *un- ion -ise -ed*. This sort of structural ambiguity isn't nearly as common in English morphology as in syntax, however. Note that *un- ion* is not a possible form (because *un-* can't attach to a noun). Furthermore, although there is a prefix *un-* that can attach to verbs, it nearly always denotes a reversal of a process (e.g., *untie*), whereas the *un-* that attaches to adjectives means 'not', which is the meaning in the case of *un- ion -ise -ed*. Hence the internal structure of *un- ion -ise -ed* has to be *(un- ((ion -ise) -ed))*.

2.3 Applications of morphological processing

It is possible to use a *full-form lexicon* for English NLP: i.e., to list all the inflected forms and to treat derivational morphology as non-productive. However, when a new word has to be handled (because the lexicon is incomplete, potentially because a new word has entered the language) it is redundant to have to specify (or learn) the inflected forms as well as the stem, since the vast majority of words in English have regular morphology. So a full-form lexicon is best regarded as a form of compilation. Many other languages have many more inflectional forms, which increases the need to do morphological analysis rather than full-form listing.

Traditional IR systems use *stemming* rather than full morphological analysis. For IR, what is required is to relate forms, not to analyse them compositionally, and this can most easily be achieved by reducing all morphologically complex forms to a canonical form. Although this is referred to as stemming, the canonical form may not be the linguistic stem. The most commonly used algorithm is the *Porter stemmer*, which uses a series of simple rules to strip endings (see J&M, section 3.8) without the need for a lexicon. However, stemming does not necessarily help IR. Search engines now generally do inflectional morphology, but this can be dangerous. For instance, searching for *corpus* as well as *corpora* when given the latter as input (as some search engines sometimes do) can result in a large number of spurious results involving *Corpus Christi* and similar terms.

In most NLP applications, however, morphological analysis is a precursor to some form of parsing. In this case, the requirement is to analyse the form into a stem and affixes so that the necessary syntactic (and possibly semantic) information can be associated with it. Morphological analysis is often called *lemmatization*. For instance, for the part of speech tagging application which I will discuss in the next lecture, *mugged* would be assigned a part of speech tag which indicates it is a verb, though *mug* is ambiguous between verb and noun. For full parsing, as discussed in lectures 4 and 5, we need more detailed syntactic and semantic information. Morphological generation takes a stem and some syntactic information and returns the correct form. For some applications, there is a requirement that morphological processing is *bidirectional*: that is, can be used for analysis and generation. The finite state transducers we will look at below have this property.

2.4 Spelling rules

English morphology is essentially concatenative: i.e., we can think of words as a sequence of prefixes, stems and suffixes. Some words have irregular morphology and their inflectional forms simply have to be listed. However, in other cases, there are regular phonological or spelling changes associated with affixation. For instance, the suffix *-s* is pronounced differently when it is added to a stem which ends in *s*, *x* or *z* and the spelling reflects this with the addition of an *e* (*boxes* etc). For the purposes of this course, I'll just talk about spelling effects rather than phonological effects: these effects can be captured by *spelling rules* (also known as *orthographic rules*).

English spelling rules can be described independently of the particular stems and affixes involved, simply in terms of the affix boundary. The 'e-insertion' rule can be described as follows:

$$\varepsilon \rightarrow \mathrm{e} / \left\{ \begin{array}{c} \mathrm{s} \\ \mathrm{x} \\ \mathrm{z} \end{array} \right\} \hat{} \_ s$$

In such rules, the mapping is always given from the 'underlying' form to the surface form, the mapping is shown to the left of the slash and the context to the right, with the _ indicating the position in question. ε is used for the empty string and ^ for the affix boundary. This particular rule is read as saying that the empty string maps to 'e' in the context where it is preceded by an s,x, or z and an affix boundary and followed by an s. For instance, this maps *box^s* to *boxes*.

This rule might look as though it is written in a context sensitive grammar formalism, but actually we'll see in §2.7 that it corresponds to a finite state transducer. Because the rule is independent of the particular affix, it applies equally to the plural form of nouns and the 3rd person singular present form of verbs. Other spelling rules in English include consonant doubling (e.g., *rat*, *ratted*, though note, not **auditted*) and y/ie conversion (*party*, *parties*).[15]

2.5 Lexical requirements for morphological processing

There are three sorts of lexical information that are needed for full, high precision morphological processing:

- affixes, plus the associated information conveyed by the affix
- irregular forms, with associated information similar to that for affixes
- stems with syntactic categories (plus more detailed information if derivational morphology is to be treated as productive)

One approach to an affix lexicon is for it to consist of a pairing of affix and some encoding of the syntactic/semantic effect of the affix.[16] For instance, consider the following fragment of a suffix lexicon (we can assume there is a separate lexicon for prefixes):

```
ed PAST_VERB
ed PSP_VERB
s PLURAL_NOUN
```

Here `PAST_VERB`, `PSP_VERB` and `PLURAL_NOUN` are abbreviations for some bundle of syntactic/semantic information and form the interface between morphology and the syntax/semantics.

A lexicon of irregular forms is also needed. One approach is for this to just be a triple consisting of inflected form, 'affix information' and stem, where 'affix information' corresponds to whatever encoding is used for the regular affix. For instance:

```
began PAST_VERB begin
begun PSP_VERB begin
```

Note that this information can be used for generation as well as analysis, as can the affix lexicon.

In most cases, English irregular forms are the same for all senses of a word. For instance, *ran* is the past of *run* whether we are talking about athletes, politicians or noses. This argues for associating irregularity with particular word forms rather than particular senses, especially since compounds also tend to follow the irregular spelling, even non-productively formed ones (e.g., the plural of *dormouse* is *dormice*). However, there are exceptions: e.g., *The washing was hung/*hanged out to dry* vs *the murderer was hanged*.

Morphological analysers may also have access to a lexicon of regular stems. This is needed for high precision: e.g. to avoid analysing *corpus* as *corpu -s*, we need to know that there isn't a word *corpu*. There are also cases where historically a word was derived, but where the base form is no longer found in the language: we can avoid analysing *unkempt* as *un- kempt*, for instance, simply by not having *kempt* in the stem lexicon. Ideally this lexicon should have syntactic information: for instance, *feed* could be *fee -ed*, but since *fee* is a noun rather than a verb, this isn't a possible analysis. However, in the approach I'll assume, the morphological analyser is split into two stages. The first of these only concerns morpheme forms and returns both *fee -ed* and *feed* given the input *feed*. A second stage which is closely coupled to the syntactic analysis then rules out *fee -ed* because the affix and stem syntactic information are not compatible.

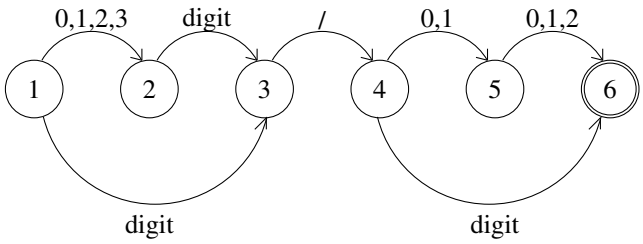
If morphology was purely concatenative, it would be very simple to write an algorithm to split off affixes. Spelling rules complicate this somewhat: in fact, it's still possible to do a reasonable job for English with ad hoc code, but a cleaner and more general approach is to use finite state techniques.

[15] Note the use of * ('star') above: this notation is used in linguistics to indicate a word or sentence which is judged (by the author, at least) to be incorrect. ? is generally used for a sentence which is questionable, or at least doesn't have the intended interpretation. # is used for a pragmatically anomalous sentence.

[16] J&M describe an alternative approach which is to make the syntactic information correspond to a level in a finite state transducer. However, at least for English, this considerably complicates the transducers.

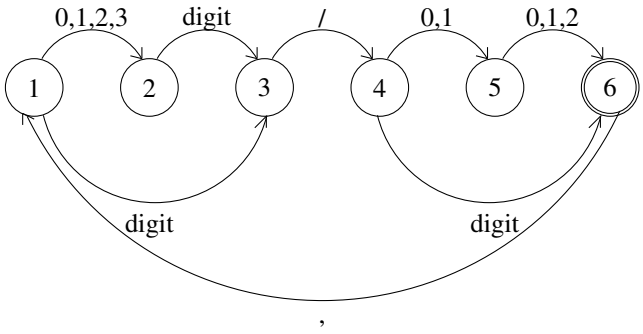
2.6 Finite state automata for recognition

The approach to spelling rules that I'll describe involves the use of finite state transducers (FSTs). Rather than jumping straight into this, I'll briefly consider the simpler finite state automata and how they can be used in a simple recogniser. Suppose we want to recognise dates (just day and month pairs) written in the format day/month. The day and the month may be expressed as one or two digits (e.g. 11/2, 1/12 etc). This format corresponds to the following simple FSA, where each character corresponds to one transition:



Accept states are shown with a double circle. This is a non-deterministic FSA: for instance, an input starting with the digit 3 will move the FSA to both state 2 and state 3. This corresponds to a *local ambiguity*: i.e., one that will be resolved by subsequent context. By convention, there must be no 'left over' characters when the system is in the final state.

To make this a bit more interesting, suppose we want to recognise a comma-separated list of such dates. The FSA, shown below, now has a cycle and can accept a sequence of indefinite length (note that this is iteration and not full recursion, however).



Both these FSAs will accept sequences which are not valid dates, such as 37/00. Conversely, if we use them to generate (random) dates, we will get some invalid output. In general, a system which generates output which is invalid is said to *overgenerate*. In fact, in many language applications, some amount of overgeneration can be tolerated, especially if we are only concerned with analysis.

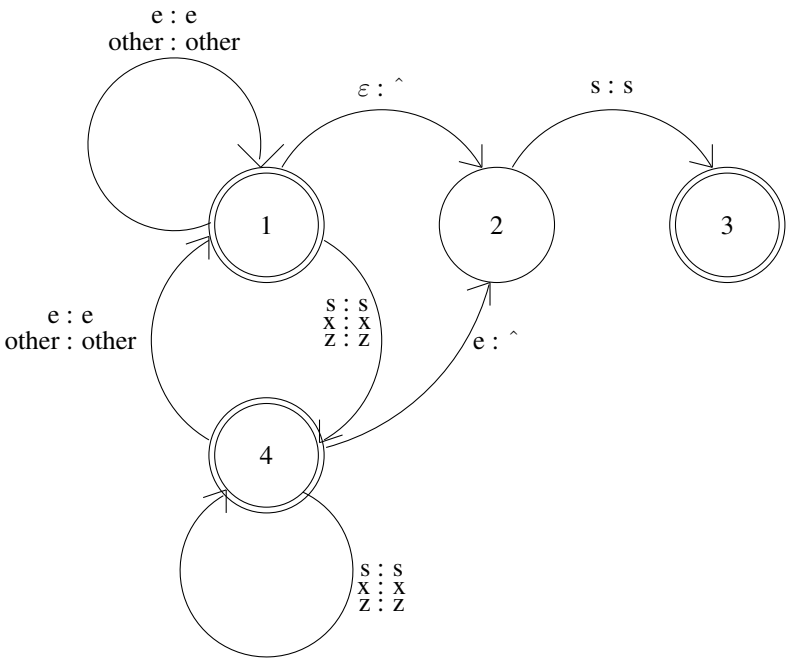
2.7 Finite state transducers

FSAs can be used to recognise particular patterns, but don't, by themselves, allow for any analysis of word forms. Hence for morphology, we use finite state transducers (FSTs) which allow the surface structure to be mapped into the list of morphemes. FSTs are useful for both analysis and generation, since the mapping is bidirectional. This approach is known as *two-level morphology*.

To illustrate two-level morphology, consider the following FST, which recognises the affix *-s* allowing for environments corresponding to the e-insertion spelling rule shown in §2.4 and repeated below.[17]

[17] Actually, I've simplified this slightly so the FST works correctly but the correspondence to the spelling rule is not exact: J&M give a more complex transducer which is an accurate reflection of the spelling rule. They also use an explicit terminating character while I prefer to rely on the 'use all the input' convention, which results in simpler rules.

$$\varepsilon \rightarrow \text{e} / \left\{ \begin{array}{c} \text{s} \\ \text{x} \\ \text{z} \end{array} \right\} \hat{} \_ s$$



Transducers map between two representations, so each transition corresponds to a pair of characters. As with the spelling rule, we use the special character 'ε' to correspond to the empty character and '^' to correspond to an affix boundary. The abbreviation 'other : other' means that any character not mentioned specifically in the FST maps to itself.[18] As with the FSA example, we assume that the FST only accepts an input if the end of the input corresponds to an accept state (i.e., no 'left-over' characters are allowed).

For instance, with this FST, the surface form *cakes* would start from 1 and go through the transitions/states (c:c) 1, (a:a) 1, (k:k) 1, (e:e) 1, (ε:^) 2, (s:s) 3 (accept, underlying *cake^s*) and also (c:c) 1, (a:a) 1, (k:k) 1, (e:e) 1, (s:s) 4 (accept, underlying *cakes*). 'd o g s' maps to 'd o g ^ s', 'f o x e s' maps to 'f o x ^ s' and to 'f o x e ^ s', and 'b u z z e s' maps to 'b u z z ^ s' and 'b u z z e ^ s'.[19] When the transducer is run in analysis mode, this means the system can detect an affix boundary (and hence look up the stem and the affix in the appropriate lexicons). In generation mode, it can construct the correct string. This FST is non-deterministic.

Similar FSTs can be written for the other spelling rules for English (although to do consonant doubling correctly, information about stress and syllable boundaries is required and there are also differences between British and American spelling conventions which complicate matters). Morphology systems are usually implemented so that there is one FST per spelling rule and these operate in parallel.

One issue with this use of FSTs is that they do not allow for any internal structure of the word form. For instance, we can produce a set of FSTs which will result in *unionised* being mapped into *un^ion^ise^ed*, but as we've seen, the affixes actually have to be applied in the right order and the bracketing isn't modelled by the FSTs.

2.8 Some other uses of finite state techniques in NLP

- Grammars for simple spoken dialogue systems. Finite state techniques are not adequate to model grammars of natural languages: I'll discuss this a little in §4.10. However, for very simple spoken dialogue systems, a finite-

[18] The solution notes for the 2003 FST question are slightly wrong in that they should have y : y as well as other : other on one transition.

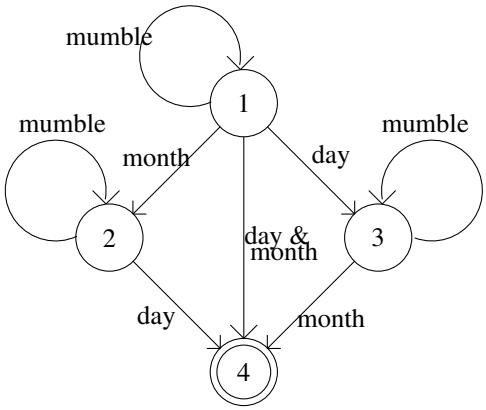
[19] In all cases they also map to themselves: e.g., 'b u z z e s' maps to 'b u z z e s' without the affix marker: this is necessary because words ending in 's' and 'es' are not always inflected forms. e.g., *Moses*

state grammar may be adequate. More complex grammars can be written as context free grammars (CFGs) and compiled into finite state approximations.

- Partial grammars for named entity recognition (briefly discussed in §4.10).
- Dialogue models for spoken dialogue systems (SDS). SDS use dialogue models for a variety of purposes: including controlling the way that the information acquired from the user is instantiated (e.g., the slots that are filled in an underlying database) and limiting the vocabulary to achieve higher recognition rates. FSAs can be used to record possible transitions between states in a simple dialogue. For instance, consider the problem of obtaining a date expressed as a day and a month from a user. There are four possible states, corresponding to the user input recognised so far:

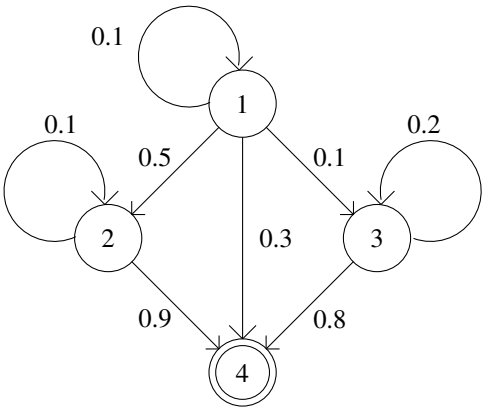
  1. No information. System prompts for month and day.
  2. Month only is known. System prompts for day.
  3. Day only is known. System prompts for month.
  4. Month and day known.

  The FSA is shown below. The loops that stay in a single state correspond to user responses that aren't recognised as containing the required information (*mumble* is the term generally used for an unrecognised input).



2.9 Probabilistic FSAs

In many cases, it is useful to augment the FSA with information about transition probabilities. For instance, in the SDS system described above, it is more likely that a user will specify a month alone than a day alone. A probabilistic FSA for the SDS is shown below. Note that the probabilities on the outgoing arcs from each state must sum to 1.



2.10 Further reading

Chapters 2 and 3 of J&M. Much of Chapter 2 should be familiar from other courses in the CST. Chapter 3 uses more elaborate transducers than I've discussed. See Bender (2013) for a much broader discussion of morphology.

3 Lecture 3: Prediction and part-of-speech tagging

This lecture introduces some simple statistical techniques and illustrates their use in NLP for prediction of words and part-of-speech categories. It starts with a discussion of corpora, then introduces word prediction. Word prediction can be seen as a way of (crudely) modelling some syntactic information (i.e., word order). Similar statistical techniques can also be used to discover parts of speech for uses of words in a corpus. The lecture concludes with some discussion of evaluation.

3.1 Corpora

A *corpus* (corpora is the plural) is simply a body of text that has been collected for some purpose. A *balanced corpus* contains texts which represent different genres (newspapers, fiction, textbooks, parliamentary reports, cooking recipes, scientific papers etc etc): early examples were the Brown corpus (US English: 1960s) and the Lancaster-Oslo-Bergen (LOB) corpus (British English: 1970s) which are each about 1 million words: the more recent British National Corpus (BNC: 1990s) contains approximately 100 million words, including about 10 million words of spoken English. Corpora are important for many types of linguistic research, although mainstream linguists in the past mostly dismissed their use in favour of reliance on intuitive judgements about whether or not an utterance is grammatical (a corpus can only (directly) provide positive evidence about grammaticality). However, many linguists do now use corpora. Corpora are essential for most modern NLP research, though NLP researchers have often used newspaper text (particularly the Wall Street Journal) or text extracted from webpages rather than balanced corpora. Distributed corpora are often annotated in some way: the most important type of annotation for NLP is part-of-speech tagging (POS tagging), which I'll discuss further below. Corpora may also be collected for a specific task. For instance, when implementing an email answering application, it is essential to collect samples of representative emails. For interface applications in particular, collecting a corpus requires a simulation of the actual application: this has often been done by a *Wizard of Oz* experiment, where a human pretends to be a computer.

Corpora are needed in NLP for two reasons. Firstly, we have to evaluate algorithms on real language: corpora are required for this purpose for any style of NLP. Secondly, corpora provide the data source for many machine-learning approaches.

3.2 Prediction

The essential idea of prediction is that, given a sequence of words, we want to determine what's most likely to come next. There are a number of reasons to want to do this, but the most interesting is that some machine learning systems can be trained using prediction on general text corpora in a way that also makes them useful on other tasks where there is limited training data. This will be discussed later in the course. Prediction is also a fundamental part of human language understanding.

In terms of direct use in applications, the most important use of prediction is as a form of *language modelling* for automatic speech recognition (ASR). Speech recognisers cannot accurately determine a word from the sound signal for that word alone, and they cannot reliably tell where each word in an utterance starts and finishes. For instance, *have an ice Dave*, *heaven ice day* and *have a nice day* could easily be confused. [20] For traditional ASR, an initial signal processing phase produces a lattice of hypotheses of the words uttered, which are then ranked and filtered using the probabilities of the possible sequences according to the language model. The language models which were traditionally most effective work on the basis of *n-grams* (a type of *Markov chain*), where the sequence of the prior $n-1$ words is used to derive a probability for the next. Trigram models use the preceding 2 words, bigram models the preceding word and unigram models use no context at all, but simply work on the basis of individual word probabilities. Bigrams are discussed below, though I won't go into details of exactly how they are used in speech recognition.

Word prediction has been used for decades in communication aids: i.e., systems for people who can't speak because of some form of disability. People who use text-to-speech systems to talk because of a non-linguistic disability usually have some form of general motor impairment which also restricts their ability to type at normal rates (stroke, ALS,

[20] In fact, although humans are better at doing this than speech recognisers, we also need context to recognise words, especially words like *the* and *a*. If a recording is made of normal, fluently spoken, speech and the segments corresponding to *the* and *a* are presented to a human subject in isolation, it's generally not possible for them to tell the difference.

cerebral palsy etc). Often they use alternative input devices, such as adapted keyboards, puffer switches, mouth sticks or eye trackers. Generally such users can only construct text at a few words a minute, which is too slow for anything like normal communication to be possible (normal speech is around 150 words per minute). As a partial aid, a word prediction system is sometimes helpful: this gives a list of candidate words that changes as the initial letters are entered by the user. The user chooses the desired word from a menu when it appears. The main difficulty with using statistical prediction models in such applications is in finding enough data: to be useful, the model really has to be trained on an individual speaker's output, but of course very little of this is likely to be available. Training a conversational aid on newspaper text can be worse than using a unigram model from the user's own data. Of course, predictive text is now commonplace on mobile devices, where users are disabled by not having a proper keyboard.

Prediction is important in estimation of entropy, including estimations of the entropy of English. The notion of entropy is important in language modelling because it gives a metric for the difficulty of the prediction problem. For instance, speech recognition is vastly easier in situations where the speaker is only saying two easily distinguishable words (e.g., when a dialogue system prompts by saying *answer 'yes' or 'no'*) than when the vocabulary is unlimited: measurements of entropy can quantify this, but won't be discussed further in this course.

Other applications for prediction include handwriting recognition, spelling correction and text segmentation for languages such as Chinese, which are conventionally written without explicit word boundaries. A form of prediction is used to model the target language in statistical MT systems, and can also be used to rank the output of grammar-based language realizers in symbolic MT or other language generation applications. Some approaches to word sense disambiguation can also be treated as a form of prediction.

3.3 bigrams

A bigram model assigns a probability to a word based on the previous word alone: i.e., $P(w_n|w_{n-1})$ (the probability of w_n conditional on w_{n-1}) where w_n is the nth word in some string. For application to communication aids, we are simply concerned with predicting the next word: once the user has made their choice, the word can't be changed. However, for speech recognition and similar applications, we require the probability of some string of words $P(w_1^n)$ which is approximated by the product of the bigram probabilities:

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

This assumes independence of the individual probabilities, which is clearly wrong, but the approximation nevertheless works reasonably well. Note that, although the n-gram probabilities are based only on the preceding words, the effect of this combination of probabilities is that the choice between possibilities at any point is sensitive to both preceding and following contexts. For instance, the decision between *a* and *the* in a lattice may be influenced by the following noun, which is a much better predictor than the words before the determiner.

We acquire these probabilities from a corpus. The bigram probabilities are given as:

$$\frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

i.e., the count of a particular bigram, normalised by dividing by the total number of bigrams starting with the same word (which is equivalent to the total number of occurrences of that word, except in the case of the last token, a complication which can be ignored for a reasonable size of corpus). For example, suppose we have the following tiny corpus of utterances:

⟨s⟩ good morning ⟨/s⟩ ⟨s⟩ good afternoon ⟨/s⟩ ⟨s⟩ good afternoon ⟨/s⟩ ⟨s⟩ it is very good ⟨/s⟩ ⟨s⟩ it is good ⟨/s⟩

I have used the symbol ⟨s⟩ to indicate the beginning of the utterance and ⟨/s⟩ to indicate the end.

sequence	count	bigram probability
<s>	5	
<s> good	3	.6
<s> it	2	.4

```
good              5
good morning      1       .2
good afternoon    2       .4
good </s>         2       .4
morning           1
morning </s>      1       1
afternoon         2
afternoon </s>    2       1
it                2
it is             2       1
is                2
is very           1       .5
is good           1       .5
very              1
very good         1       1
</s>              5
</s><s>           4       1
```

This yields a probability of 0.24 for the string '⟨s⟩ good ⟨/s⟩' and also for '⟨s⟩ good afternoon ⟨/s⟩'.

Notice that we can regard bigrams as comprising a simple deterministic weighted FSA. The *Viterbi algorithm*, a dynamic programming technique for efficiently applying n-grams in speech recognition and other applications to find the highest probability sequence (or sequences), is usually described in terms of an FSA.

With the approach described, the probability of '⟨s⟩ very good ⟨/s⟩' based on this toy corpus is 0: the conditional probability of 'very' given '⟨s⟩' is 0 since there are no examples of this in the training data. Even for realistically sized corpora, zero probabilities are problematic: we will never have enough data to ensure that we will see all possible events and so we don't want to rule out unseen events entirely. To allow for *sparse data* we have to use *smoothing*, which simply means that we make some assumption about the 'real' probability of unseen or very infrequently seen events and distribute that probability appropriately. A common approach is simply to add one to all counts: this is *add-one smoothing* which is not sound theoretically, but is simple to implement. A better approach in the case of bigrams is to *backoff* to the unigram probabilities: i.e., to distribute the unseen probability mass so that it is proportional to the unigram probabilities. This sort of estimation is extremely important to get good results from n-gram techniques, but is not discussed further here.

3.4 Part of speech tagging

Sometimes we are interested in a form of prediction that involves assigning classes to items in a sequence rather than predicting the next item. One important application is to part-of-speech tagging (POS tagging), where the words in a corpus are associated with a tag indicating some syntactic information that applies to that particular use of the word. For instance, consider the example sentence below:

> They can fish.

This has two readings: one (the most likely) about ability to fish and other about putting fish in cans. *fish* is ambiguous between a singular noun, plural noun and a verb, while *can* is ambiguous between singular noun, verb (the 'put into cans' use) and modal verb (the 'possibility' use). However, *they* is unambiguously a pronoun. (I am ignoring some less likely possibilities, such as proper names.) These distinctions could be indicated by POS tags:

> They_pronoun can_modal fish_verb.
> They_pronoun can_verb fish_plural-noun.

In fact, much less mnemonic tag names are used in the standard tagsets for POS tagging. For the examples in this lecture I'll use the CLAWS 5 (C5) tagset which is given in full in Figure 5.9 in J&M. The tags needed for the example above are:

```
NN1 singular noun
NN2 plural noun
PNP personal pronoun
VM0 modal auxiliary verb
VVB base form of verb (except infinitive)
VVI infinitive form of verb (i.e. occurs with 'to' and in similar contexts)
```

The lexicon which associates the words with the tags includes:

```
they PNP
can VM0 VVB VVI NN1
fish NN1 NN2 VVB VVI
```

A POS tagger resolves the lexical ambiguities to give the most likely set of tags for the sentence, i.e., for this example:

They_PNP can_VM0 fish_VVI ._PUN

Note the tag for the full stop: punctuation is treated as unambiguous.

The other syntactically possible reading mentioned above corresponds to the following using C5 tags:

They_PNP can_VVB fish_NN2 ._PUN

However, POS taggers (unlike full parsers) don't attempt to produce globally coherent analyses. Thus a POS tagger might also return:

They_PNP can_VM0 fish_NN2 ._PUN

despite the fact that this doesn't correspond to a possible reading of the sentence.

POS tagging can be regarded as a form of very basic, coarse-grained, sense disambiguation. It is useful as a way of annotating a corpus because it makes it easier to extract some types of information (for linguistic research or NLP experiments). It also acts as a basis for more complex forms of annotation. Named entity recognisers (discussed in lecture 4) are generally run on POS-tagged data. As mentioned in lecture 1, POS taggers are sometimes run as preprocessors to full parsing, since this can cut down the search space to be considered by the parser. They can also be used as part of a method for dealing with words which are not in the parser's lexicon (unknown words).

3.5 Stochastic POS tagging using Hidden Markov Models

One form of POS tagging uses a technique known as *Hidden Markov Modelling* (HMM). It involves an n-gram technique, but in this case the n-grams are sequences of POS tags rather than of words. The most common approaches depend on a small amount of manually tagged *training data* from which POS n-grams can be extracted.[21] I'll illustrate this with respect to another trivial corpus:

They used to can fish in those towns. But now few people fish in these areas.

This might be tagged as follows:

```
They_PNP used_VVD to_TO0 can_VVI fish_NN2 in_PRP those_DT0 towns_NN2 ._PUN
But_CJC now_AV0 few_DT0 people_NN2 fish_VVB in_PRP these_DT0 areas_NN2 ._PUN
```

This yields the following counts and probabilities:

[21] It is possible to build POS taggers that work without a hand-tagged corpus, but they don't perform as well as a system trained on even a very small manually-tagged corpus and they still require a lexicon associating possible tags with words. Completely unsupervised approaches also exist, where no lexicon is used, but the categories induced do not correspond to any standard tagset.

sequence	count	bigram probability
AV0	1	
AV0 DT0	1	1
CJC	1	
CJC AV0	1	1
DT0	3	
DT0 NN2	3	1
NN2	4	
NN2 PRP	1	0.25
NN2 PUN	2	0.5
NN2 VVB	1	0.25
PNP	1	
PNP VVD	1	1
PRP	1	
PRP DT0	2	1
PUN	1	
PUN CJC	1	1
TO0	1	
TO0 VVI	1	1
VVB	1	
VVB PRP	1	1
VVD	1	
VVD TO0	1	1
VVI	1	
VVI NN2	1	1

I've used the correct PUN CJC probability, allowing for the final PUN. We also obtain a lexicon from the tagged data:

word	tag	count	word	tag	count
they	PNP	1	towns	NN2	1
used	VVD	1	.	PUN	1
to	TO0	1	but	CJC	1
can	VVI	1	now	AV0	1
fish	NN2	1	few	DT0	1
	VVB	1	people	NN2	1
in	PRP	2	these	DT0	1
those	DT0	1	areas	NN2	1

The idea of stochastic POS tagging is that the tag can be assigned based on consideration of the lexical probability (how likely it is that the word has that tag), plus the sequence of prior tags (for a bigram model, the immediately prior tag). This is more complicated than prediction because we have to take into account both words and tags.

We wish to produce a sequence of tags which have the maximum probability given a sequence of words. I will follow J&M's notation: the hat, $\hat{}$, means "estimate of"; $\hat{t}_1^n$ means "estimate of the sequence of n tags"; $\underset{x}{\operatorname{argmax}} f(x)$ means "the x such that $f(x)$ is maximized". Hence:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n)$$

We can't estimate this directly (mini-exercise: explain why not). By Bayes theorem:

$$P(t_1^n | w_1^n) = \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)}$$

Since we're looking at assigning tags to a particular sequence of words, $P(w_1^n)$ is constant, so for a relative measure of probability we can use:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(w_1^n | t_1^n) P(t_1^n)$$

We now have to estimate $P(t_1^n)$ and $P(w_1^n|t_1^n)$. If we make the bigram assumption, then the probability of a tag depends on the previous tag, hence the tag sequence is estimated as a product of the probabilities:

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1})$$

We will also assume that the probability of the word is independent of the words and tags around it and depends only on its own tag:

$$P(w_1^n|t_1^n) \approx \prod_{i=1}^{n} P(w_i|t_i)$$

These values can be estimated from the corpus frequencies. So our final equation for the HMM POS tagger using bigrams is:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

Note that we end up multiplying $P(t_i|t_{i-1})$ with $P(w_i|t_i)$ (the probability of the word given the tag) rather than $P(t_i|w_i)$ (the probability of the tag given the word). For instance, if we're trying to choose between the tags NN2 and VVB for *fish* in the sentence *they fish*, we calculate P(NN2|PNP), P(fish|NN2), P(VVB|PNP) and P(fish|VVB) (assuming PNP is the only possible tag for *they*).

As the equation above indicates, in order to POS tag a sentence, we maximise the overall tag sequence probability (again, this can be implemented efficiently using the Viterbi algorithm). So a tag which has high probability considering its individual bigram estimate will not be chosen if it does not form part of the highest probability path. For example, consider:
they_PNP can_VVB fish_NN2
they_PNP can_VM0 fish_VVI
The product of P(VVI|VM0) and P(fish|VVI) may be lower than that of P(NN2|VVB) and P(fish|NN2) but the overall probability depends also on P(can|VVB) versus P(can|VM0) and the latter (modal) use has much higher frequency in a balanced corpus.

In fact, POS taggers generally use trigrams rather than bigrams — the relevant equations are given in J&M, 5.5.4. As with word prediction, backoff (to bigrams) and smoothing are crucial for reasonable performance because of sparse data.

When a POS tagger sees a word which was not in its training data, we need some way of assigning possible tags to the word. One approach is simply to use all possible *open class* tags, with probabilities based on the unigram probabilities of those tags.[22] A better approach is to use a morphological analyser (without a lexicon) to restrict the candidates: e.g., words ending in *-ed* are likely to be VVD (simple past) or VVN (past participle), but can't be VVG (-ing form).

3.6 Evaluation of POS tagging

POS tagging algorithms are evaluated in terms of percentage of correct tags. The standard assumption is that every word should be tagged with exactly one tag, which is scored as correct or incorrect: there are no marks for near misses. Generally there are some words which can be tagged in only one way, so are automatically counted as correct. Punctuation is generally given an unambiguous tag. Success rates of about 97% are possible for English POS tagging (performance seems to have reached a plateau, probably partly because of errors in the manually-tagged corpora) but the baseline of choosing the most common tag based on the training set often gives about 90% accuracy.[23] Some POS taggers return multiple tags in cases where more than one tag has a similar probability.

Increasing the size of the tagset does not necessarily result in decreased performance: some additional tags could be assigned more-or-less unambiguously and more fine-grained tags can increase performance. For instance, suppose we wanted to distinguish between present tense verbs according to whether they were 1st, 2nd or 3rd person. With the C5

[22] Open class words are ones for which we can never give a complete list for a living language, since words are always being invented: i.e., verbs, nouns, adjectives and adverbs. Other words (prepositions, determiners, conjunctions and so on) are considered closed class.

[23] Accuracy differs between languages mainly because of differences in morphology: Japanese POS-tagging is almost deterministic but accuracy for Turkish is only around 90%.

tagset, and the stochastic tagger described, this would be impossible to do with high accuracy, because all pronouns are tagged PRP, hence they provide no discriminating power. On the other hand, if we tagged *I* and *we* as PRP1, *you* as PRP2 and so on, the n-gram approach would allow some discrimination. In general, predicting on the basis of classes means we have less of a sparse data problem than when predicting on the basis of words, but we have less discriminating power. There is also something of a trade-off between the utility of a set of tags and their effectiveness in POS tagging. For instance, C5 assigns separate tags for the different forms of *be*, which is redundant for many purposes, but helps make distinctions between other tags in tagging models such as the HMM described here where the context is given by a tag sequence alone (i.e., rather than considering words prior to the current one).

POS tagging exemplifies some general issues in NLP evaluation:

Training data and test data The assumption in NLP is always that a system should work on novel data, therefore test data must be kept unseen.

For machine learning approaches, such as stochastic POS tagging, the usual technique is to split a data set into 90% training and 10% test data. Care needs to be taken that the test data is representative.

For an approach that relies on significant hand-coding, the test data should be literally unseen by the researchers. Development cycles involve looking at some initial data, developing the algorithm, testing on unseen data, revising the algorithm and testing on a new batch of data. The seen data is kept for regression testing.

Baselines Evaluation should be reported with respect to a baseline, which is normally what could be achieved with a very basic approach, given the same training data. For instance, a baseline for POS tagging with training data is to choose the most common tag for a particular word on the basis of the training data (and to simply choose the most frequent tag of all for unseen words).

Ceiling It is often useful to try and compute some sort of ceiling for the performance of an application. This is usually taken to be human performance on that task, where the ceiling is the percentage agreement found between two annotators (*interannotator agreement*). For POS tagging, this has been reported as 96% (which makes existing POS taggers look impressive since some perform at higher accuracy). However this raises lots of questions: relatively untrained human annotators working independently often have quite low agreement, but trained annotators discussing results can achieve much higher performance (approaching 100% for POS tagging). Human performance varies considerably between individuals. Fatigue can cause errors, even with very experienced annotators. In any case, human performance may not be a realistic ceiling on relatively unnatural tasks, such as POS tagging.

Error analysis The error rate on a particular problem will be distributed very unevenly. For instance, a POS tagger will never confuse the tag PUN with the tag VVN (past participle), but might confuse VVN with AJ0 (adjective) because there's a systematic ambiguity for many forms (e.g., *given*). For a particular application, some errors may be more important than others. For instance, if one is looking for relatively low frequency cases of denominal verbs (that is verbs derived from nouns — e.g., *canoe*, *tango*, *fork* used as verbs), then POS tagging is not directly useful in general, because a verbal use without a characteristic affix is likely to be mistagged. This makes POS-tagging less useful for lexicographers, who are often specifically interested in finding examples of unusual word uses. Similarly, in text categorisation, some errors are more important than others: e.g. treating an incoming order for an expensive product as junk email is a much worse error than the converse.

Reproducibility If at all possible, evaluation should be done on a generally available corpus so that other researchers can replicate the experiments.

3.7 Further reading

N-grams are described in Chapter 4 of J&M, POS tagging in Chapter 5. The description in the second edition is considerably clearly than that in the first edition.

4 Lecture 4: Context-free grammars and parsing.

In this lecture, I'll discuss syntax in a way which is much closer to the standard notions in formal linguistics than POS-tagging is. To start with, I'll briefly motivate the idea of a generative grammar in linguistics, review the notion of a context-free grammar and then show a context-free grammar for a tiny fragment of English. We'll then see how context free grammars can be used to implement parsers, and discuss chart parsing, which allows efficient processing of strings containing a high degree of ambiguity. Finally we'll briefly touch on probabilistic context-free approaches.

4.1 Generative grammar

Since Chomsky's work in the 1950s, much work in formal linguistics has been concerned with the notion of a *generative grammar* — i.e., a formally specified grammar that can generate all and only the acceptable sentences of a natural language. It's important to realise that nobody has actually written a complete grammar of this type for any living natural language:[24] what most linguists are really interested in is the principles that underlie such grammars, especially to the extent that they apply to all natural languages. NLP researchers, on the other hand, are at least sometimes interested in actually building and using large-scale detailed grammars.

The formalisms which are of interest to us for modelling syntax assign internal structure to the strings of a language, which may be represented by bracketing. We already saw some evidence of this in derivational morphology (the *unionised* example), but here we are concerned with the structure of phrases. For instance, the sentence:

the big dog slept

can be bracketed

((the (big dog)) slept)

The phrase, *big dog*, is an example of a *constituent* (i.e. something that is enclosed in a pair of brackets): *the big dog* is also a constituent, but *the big* and *dog slept* are not. Constituent structure is generally justified by arguments about substitution which I won't go into here: J&M discuss this briefly, as does Bender (2013) but see an introductory syntax book for a full discussion. In this course, I will simply give bracketed structures and hope that the constituents make sense intuitively, rather than trying to justify them.

Two grammars are said to be *weakly-equivalent* if they generate the same strings. Two grammars are *strongly-equivalent* if they assign the same structure to all strings they generate.

In most, but not all, approaches, the internal structures are given labels.[25] For instance, *the big dog* is a *noun phrase* (abbreviated NP), *slept*, *slept in the park* and *licked Sandy* are *verb phrase*s (VPs). The labels such as NP and VP correspond to non-terminal symbols in a grammar. In this lecture, I'll discuss the use of simple context-free grammars for language description.

4.2 Context free grammars

The idea of a context-free grammar (CFG) should be familiar from formal language theory. A CFG has four components, described here as they apply to grammars of natural languages:

1. a set of non-terminal symbols (e.g., S, VP), conventionally written in uppercase;
2. a set of terminal symbols (i.e., the words), conventionally written in lowercase;
3. a set of rules (productions), where the left hand side (the mother) is a single non-terminal and the right hand side is a sequence of one or more non-terminal or terminal symbols (the daughters);
4. a start symbol, conventionally S, which is a member of the set of non-terminal symbols.

[24] There are grammars which cover all recorded utterances of extinct languages.

[25] The alternative *dependency* approach will be discussed in lecture 5.

The formal description of a CFG generally allows productions with an empty right hand side (e.g., Det $\rightarrow \varepsilon$). It is convenient to exclude these however, since they complicate parsing algorithms, and a weakly-equivalent grammar can always be constructed that disallows such *empty productions*.

A grammar in which all nonterminal daughters are the leftmost daughter in a rule (i.e., where all rules are of the form $X \rightarrow Ya*$), is said to be *left-associative*. A grammar where all the nonterminals are rightmost is *right-associative*. Such grammars are weakly-equivalent to regular grammars (i.e., grammars that can be implemented by FSAs), but natural languages seem to require more expressive power than this (see §4.10).

4.3 A simple CFG for a fragment of English

The following tiny fragment is intended to illustrate some of the properties of CFGs so that we can discuss parsing. It has some serious deficiencies as a representation of even this fragment, which I'll ignore for now, though we'll see some of them at the end of the lecture. Notice that for this fragment there is no distinction between main verb *can* and the modal verb *can*.

```
S -> NP VP
VP -> VP PP
VP -> V
VP -> V NP
VP -> V VP
NP -> NP PP
PP -> P NP
;;; lexicon
V -> can
V -> fish
NP -> fish
NP -> rivers
NP -> pools
NP -> December
NP -> Scotland
NP -> it
NP -> they
P -> in
```

The rules with terminal symbols on the right hand side correspond to the lexicon. Here and below, comments are preceded by `;;;`

Here are some strings which this grammar generates, along with their bracketings:

they fish
(S (NP they) (VP (V fish)))

they can fish
(S (NP they) (VP (V can) (VP (V fish))))
;;; the modal verb 'are able to' reading
(S (NP they) (VP (V can) (NP fish)))
;;; the less plausible, put fish in cans, reading

they fish in rivers
(S (NP they) (VP (VP (V fish)) (PP (P in) (NP rivers))))

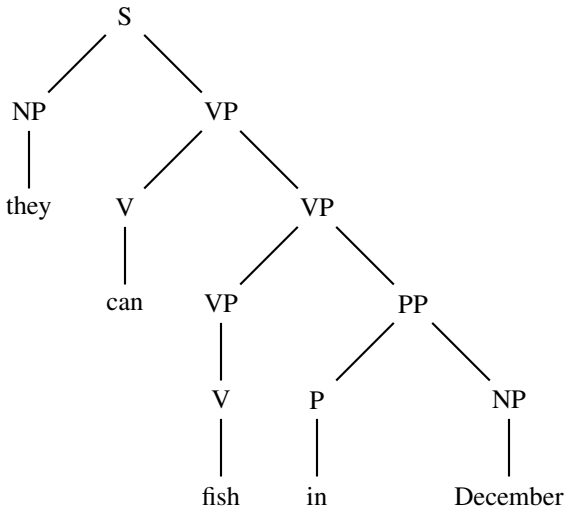
they fish in rivers in December
(S (NP they) (VP (VP (V fish)) (PP (P in) (NP (NP rivers) (PP (P in) (NP December))))))
;;; i.e. the implausible reading where the rivers are in December
;;; (cf rivers in Scotland)
(S (NP they) (VP (VP (VP (V fish)) (PP (P in) (NP rivers))) (PP (P in) (NP December))))
;;; i.e. the fishing is done in December

One important thing to notice about these examples is that there's lots of potential for ambiguity. In the *they can fish* example, this is due to *lexical ambiguity* (it arises from the dual lexical entries for *fish*), but the last example demonstrates purely *structural ambiguity*. In this case, the ambiguity arises from the two possible *attachments* of the prepositional phrase (PP) *in December*: it can attach to the NP (*rivers*) or to the VP. These attachments correspond to different semantics, as indicated by the glosses. PP attachment ambiguities are a major headache in parsing, since sequences of four or more PPs are common in real texts and the number of readings increases as the Catalan series, which is exponential. Other phenomena have similar properties: for instance, compound nouns (e.g. *long-stay car park shuttle bus*). Humans disambiguate such attachments as they hear a sentence, but they're relying on the meaning in context to do this, in a way we cannot currently emulate, except when the sentences are restricted to a very limited domain.

Notice that *fish* could have been entered in the lexicon directly as a VP, but that this would cause problems if we were doing inflectional morphology, because we want to say that suffixes like *-ed* apply to Vs. Making *rivers* etc NPs rather than nouns is a simplification I've adopted here just to keep the example grammar smaller.

4.4 Parse trees

Parse trees are equivalent to bracketed structures, but are easier to read for complex cases. A parse tree and bracketed structure for one reading of *they can fish in December* is shown below. The correspondence should be obvious.




```
(S (NP they)
   (VP (V can)
       (VP (VP (V fish))
           (PP (P in)
               (NP December)))))
```

4.5 Chart parsing

In order to parse with reasonable efficiency, we need to keep a record of the rules that we have applied so that we don't have to backtrack and redo work that we've done before. This works for parsing with CFGs because the rules are independent of their context: a VP can always expand as a V and an NP regardless of whether or not it was preceded by an NP or a V, for instance. (In some cases we may be able to apply techniques that look at the context to cut down the search space, because we can tell that a particular rule application is never going to be part of a sentence, but this is strictly a filter: we're never going to get incorrect results by reusing partial structures.) This record keeping strategy is an application of dynamic programming/memoization which is used in processing formal languages too. In NLP the data structure used for recording partial results is generally known as a *chart* and algorithms for parsing using such structures are referred to as *chart parsers*.[26] Chart parsing strategies are designed to be *complete*: that is, the chart

[26]Natural languages have vastly higher degrees of ambiguity than programming languages: chart parsing is well-suited to this.

parser will find all valid analyses according to a grammar (though there are some minor caveats e.g., concerning rules which can apply to their own output).

A chart is a collection of *edges*, usually implemented as a vector of edges, indexed by edge identifiers. In the simplest version of chart parsing, each edge records a rule application and has the following structure:

[*id*,*left_vertex*, *right_vertex*,*mother_category*, *daughters*]

A vertex is an integer representing a point in the input string, as illustrated below:

```
. they . can . fish .
0      1     2      3
```

mother_category refers to the rule that has been applied to create the edge. *daughters* is a list of the edges that acted as the daughters for this particular rule application: it is there purely for record keeping so that the output of parsing can be a labelled bracketing.

For instance, the following edges would be among those found on the chart after a complete parse of *they can fish* according to the grammar given above (id numbering is arbitrary):

```
id  left   right  mother   daughters
3    1       2      V        (can)
4    2       3      NP       (fish)
5    2       3      V        (fish)
6    2       3      VP       (5)
7    1       3      VP       (3 6)
8    1       3      VP       (3 4)
```

The daughters for the terminal rule applications are simply the input word strings.

Note that local ambiguities correspond to situations where a particular span has more than one associated edge. We'll see below that we can *pack* structures so that we never have two edges with the same category and the same span, but we'll ignore this for the moment (see §4.8). Also, in this chart we're only recording complete rule applications: this is *passive* chart parsing.

4.6 A bottom-up passive chart parser

The following pseudo-code sketch is for a very simple chart parser. Informally, it proceeds by adding the next word (in left to right order), and adding each lexical category possible for that word, doing everything it can immediately after each lexical category is added. The main function is **Add new edge** which is called for each word in the input going left to right. **Add new edge** recursively scans backwards looking for other daughters.

Parse:
  Initialise the chart (i.e., clear previous results)
  For each word *word* in the input sentence, let *from* be the left vertex, *to* be the right vertex and *daughters* be (*word*)
    For each category *category* that is lexically associated with *word*
      Add new edge *from*, *to*, *category*, *daughters*
  Output results for all spanning edges
  (i.e., ones that cover the entire input and which have a mother corresponding to the root category)

Add new edge *from*, *to*, *category*, *daughters*:
  Put edge in chart: [*id*,*from*,*to*, *category*,*daughters*]
  For each *rule* in the grammar of form *lhs* –> $cat_1 \ldots cat_{n-1}$,*category*
    Find set of lists of contiguous edges $[id_1,from_1,to_1, cat_1,daughters_1] \ldots [id_{n-1},from_{n-1},from, cat_{n-1},daughters_{n-1}]$
    (such that $to_1 = from_2$ etc)
    (i.e., find all edges that match a rule)
    For each list of edges, **Add new edge** $from_1$, *to*, *lhs*, ($id_1 \ldots id$)
    (i.e., apply the rule to the edges)

Notice that this means that the grammar rules are indexed by their rightmost category, and that the edges in the chart must be indexed by their *to* vertex (because we scan backward from the rightmost category). Consider:

```
. they . can . fish .
0      1     2      3
```

The following diagram shows the chart edges as they are constructed in order (when there is a choice, taking rules in a priority order according to the order they appear in the grammar):

```
id  left   right  mother   daughters
1    0       1      NP       (they)
2    1       2      V        (can)
3    1       2      VP       (2)
4    0       2      S        (1 3)
5    2       3      V        (fish)
6    2       3      VP       (5)
7    1       3      VP       (2 6)
8    0       3      S        (1 7)
9    2       3      NP       (fish)
10   1       3      VP       (2 9)
11   0       3      S        (1 10)
```

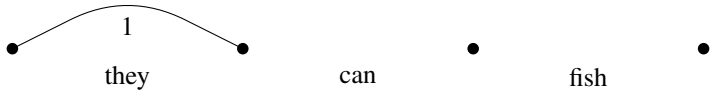
The spanning edges are 11 and 8: the output routine to give bracketed parses simply outputs a left bracket, outputs the category, recurses through each of the daughters and then outputs a right bracket. So, for instance, the output from edge 11 is:

```
(S (NP they) (VP (V can) (NP fish)))
```

This chart parsing algorithm is *complete*: it returns all possible analyses (except in the case where it does not terminate because there is a unary rule that applies to its own output).

4.7 A detailed trace of the simple chart parser

Parse
word = they
categories = NP
Add new edge 0, 1, NP, (they)

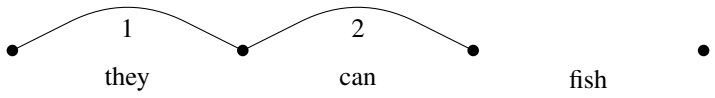


Matching grammar rules are:

```
VP -> V NP
PP -> P NP
```

No matching edges corresponding to V or P

word = can
categories = V
Add new edge 1, 2, V, (can)

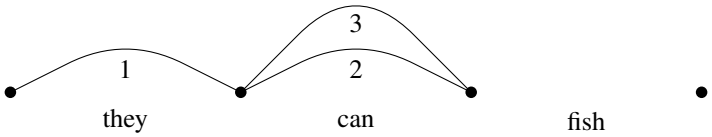


Matching grammar rules are:

```
VP -> V
```

set of edge lists $= \{(2)\}$

Add new edge 1, 2, VP, (2)

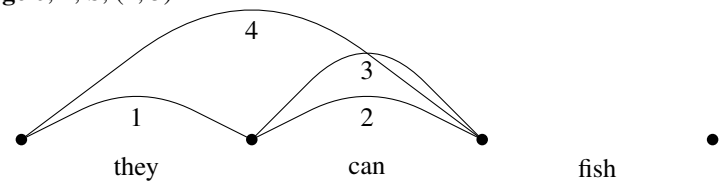


Matching grammar rules are:

```
S -> NP VP
VP -> V VP
```

set of edge lists corresponding to NP VP $= \{(1, 3)\}$

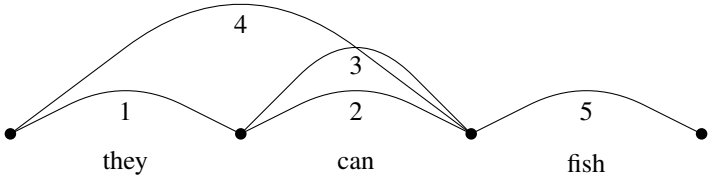
Add new edge 0, 2, S, (1, 3)



No matching grammar rules for S

No edges matching V VP

word = fish
categories = V, NP
Add new edge 2, 3, V, (fish)

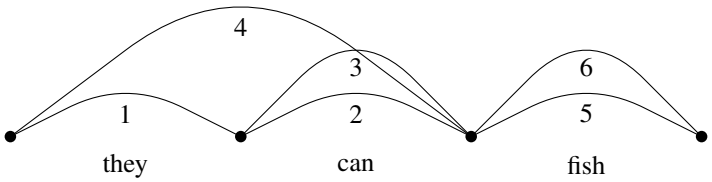


Matching grammar rules are:

```
VP -> V
```

set of edge lists $= \{(5)\}$

Add new edge 2, 3, VP, (5)

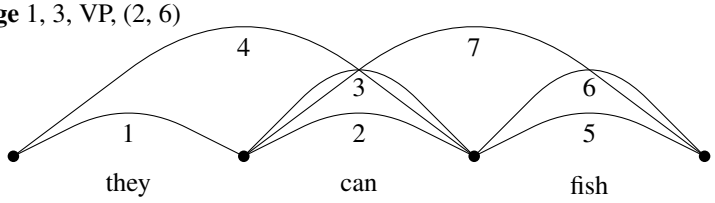


Matching grammar rules are:

```
S -> NP VP
VP -> V VP
```

No edges match NP
set of edge lists for V VP $= \{(2, 6)\}$

Add new edge 1, 3, VP, (2, 6)

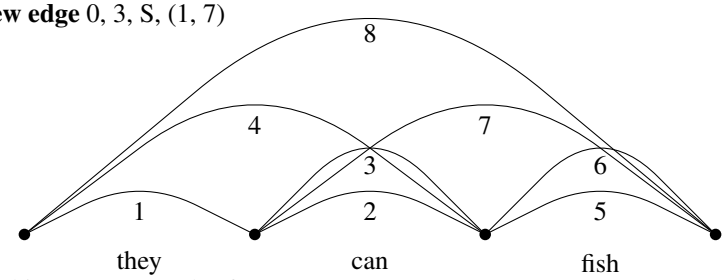


Matching grammar rules are:

```
S -> NP VP
VP -> V VP
```

set of edge lists for NP VP $= \{(1, 7)\}$

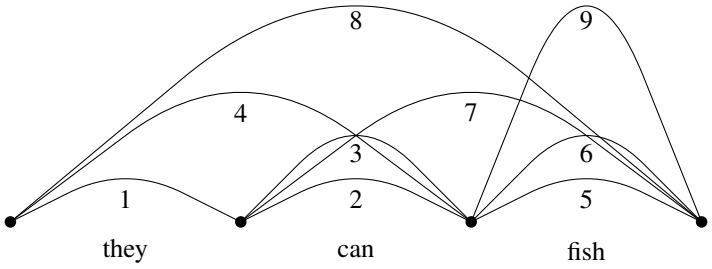
Add new edge 0, 3, S, (1, 7)



No matching grammar rules for S

No edges matching V

Add new edge 2, 3, NP, (fish)

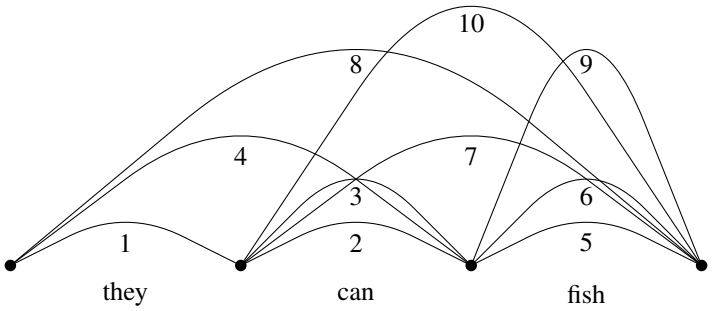


Matching grammar rules are:

```
VP -> V NP
PP -> P NP
```

set of edge lists corresponding to V NP $= \{(2, 9)\}$

Add new edge 1, 3, VP, (2, 9)

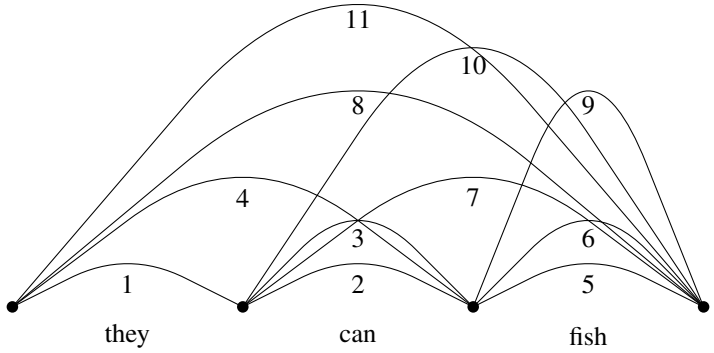


Matching grammar rules are:

```
S -> NP VP
VP -> V VP
```

set of edge lists corresponding to NP VP $= \{(1, 10)\}$

Add new edge 0, 3, S, (1, 10)



No matching grammar rules for S

No edges corresponding to V VP

No edges corresponding to P NP

No further words in input

Spanning edges are 8 and 11: Output results for 8

```
(S (NP they) (VP (V can) (VP (V fish))))
```

Output results for 11

```
(S (NP they) (VP (V can) (NP fish)))
```

4.8 Packing

The algorithm given above is exponential in the case where there are an exponential number of parses. The body of the algorithm can be modified so that it runs in cubic time, though producing the output is still exponential. The modification is simply to change the daughters value on an edge to be a set of lists of daughters and to make an equality check before adding an edge so we don't add one that's equivalent to an existing one. That is, if we are about to add an edge:

[*id*,*left_vertex*, *right_vertex*,*mother_category*, *daughters*]

and there is an existing edge:

[*id-old*,*left_vertex*, *right_vertex*,*mother_category*, *daughters-old*]

we simply modify the old edge to record the new daughters:

[*id-old*,*left_vertex*, *right_vertex*,*mother_category*, *daughters-old* $\sqcup$ *daughters*]

There is no need to recurse with this edge, because we couldn't get any new results: once we've found we can pack an edge, we always stop that part of the search. Thus packing saves computation and in fact leads to cubic time operation, though I won't go through the proof of this.

For the example above, everything proceeds as before up to edge 9:

```
id  left   right  mother   daughters
1    0       1      NP       {(they)}
2    1       2      V        {(can)}
3    1       2      VP       {(2)}
4    0       2      S        {(1 3)}
5    2       3      V        {(fish)}
6    2       3      VP       {(5)}
7    1       3      VP       {(2 6)}
8    0       3      S        {(1 7)}
9    2       3      NP       {(fish)}
```

However, rather than add edge 10, which would be:

```
10   1       3      VP       (2 9)
```

we match this with edge 7, and simply add the new daughters to that.

```
7    1       3      VP       {(2 6), (2 9)}
```

The algorithm then terminates. We only have one spanning edge (edge 8) but the display routine is more complex because we have to consider the alternative sets of daughters for edge 7. (You should go through this to convince yourself that the same results are obtained as before.) Although in this case, the amount of processing saved is small, the effects are much more important with longer sentences (consider *he believes they can fish*, for instance).

4.9 Ordering the search space

In the pseudo-code above, the order of addition of edges to the chart was determined by the recursion. In general, chart parsers make use of an *agenda* of edges, so that the next edges to be operated on are the ones that are first on the agenda. Different parsing algorithms can be implemented by making this agenda a stack or a queue, for instance.

So far, we've considered *bottom up* parsing: an alternative is *top down* parsing, where the initial edges are given by the rules whose mother corresponds to the start symbol.

Some efficiency improvements can be obtained by ordering the search space appropriately, though which version is most efficient depends on properties of the individual grammar. However, the most important reason to use an explicit agenda is when we are returning parses in some sort of priority order, corresponding to weights on different grammar rules or lexical entries.

Weights can be manually assigned to rules and lexical entries in a manually constructed grammar. However, since the beginning of the 1990s, a lot of work has been done on automatically acquiring probabilities from a corpus annotated with syntactic trees (a *treebank*), either as part of a general process of automatic grammar acquisition, or as automatically acquired additions to a manually constructed grammar. Probabilistic CFGs (PCFGs) can be defined quite straightforwardly, if the assumption is made that the probabilities of rules and lexical entries are independent of one another (of course this assumption is not correct, but the orderings given seem to work quite well in practice). The importance of this is that we rarely want to return all parses in a real application, but instead we want to return those which are top-ranked: i.e., the most likely parses. This is especially true when we consider that realistic grammars can easily return many tens of thousands of parses for sentences of quite moderate length (20 words or so). If edges are prioritised by probability, very low priority edges can be completely excluded from consideration if there is a cut-off such that we can be reasonably certain that no edges with a lower priority than the cut-off will contribute to the highest-ranked parse. Limiting the number of analyses under consideration is known as *beam search* (the analogy is that we're looking within a beam of light, corresponding to the highest probability edges). Beam search is linear rather than exponential or cubic. Just as importantly, a good priority ordering from a parser reduces the amount of work that has to be done to filter the results by whatever system is processing the parser's output.

4.10 Why can't we use FSAs to model the syntax of natural languages?

In this lecture, we started using CFGs. This raises the question of why we need this more expressive (and hence computationally expensive) formalism, rather than modelling syntax with FSAs.[27] The usual answer is that the syntax of natural languages cannot be described by an FSA, even in principle, due to the presence of *centre-embedding*, i.e. structures which map to:

$$A \rightarrow \alpha A \beta$$

and which generate grammars of the form $a^n b^n$. For instance:

the students the police arrested complained

has a centre-embedded structure. However, this is not entirely satisfactory, since humans have difficulty processing more than two levels of embedding:

? the students the police the journalists criticised arrested complained

If the recursion is finite (no matter how deep), then the strings of the language could be generated by an FSA. So it's not entirely clear whether an FSA might not suffice, despite centre embedding.

There's a fairly extensive discussion of the theoretical issues in J&M , but there are two essential points for our purposes:

1. Grammars written using finite state techniques alone may be very highly redundant, which makes them difficult to build and slow to run.
2. Without internal structure, we can't build up good semantic representations.

These are the main reasons for the use of more powerful formalisms from an NLP perspective (in the next section, I'll discuss whether simple CFGs are inadequate for similar reasons).

However, FSAs are very useful for partial grammars. In particular, for information extraction, we need to recognise *named entities*: e.g. Professor Smith, IBM, 101 Dalmatians, the White House, the Alps and so on. Although NPs are in general recursive (*the man who likes the dog which bites postmen*), relative clauses are not generally part of named entities. Also the internal structure of the names is unimportant for IE. Hence FSAs can be used, with sequences such as 'title surname', 'DT0 PNP' etc

CFGs can be automatically compiled into approximately equivalent FSAs by putting bounds on the recursion. This is particularly important in speech recognition engines.

4.11 Deficiencies in atomic category CFGs

If we consider the sample grammar in §4.3, several problems are apparent. One is that there is no account of subject-verb agreement, so, for instance, **it fish* is allowed by the grammar as well as *they fish*.[28]

We could, of course, allow for agreement by increasing the number of atomic symbols in the CFG, introducing NP-sg, NP-pl, VP-sg and VP-pl, for instance. But this approach would soon become very tedious:

```
S -> NP-sg VP-sg
S -> NP-pl VP-pl
VP-sg -> V-sg NP-sg
VP-sg -> V-sg NP-pl
```

[27] Recall that any FSA can be converted to a *regular grammar*, so grammar rules written in the format we've been using could be parsed using an FSA if they met the restrictions for a regular grammar.

[28] In English, the subject of a sentence is generally a noun phrase which comes before the verb, in contrast to the object, which follows the verb. The subject and the verb must (usually) either both have singular morphology or both have plural morphology: i.e., they must *agree*. There was also no account of *case*: this is only reflected in a few places in modern English, but **they can they* is clearly ungrammatical (as opposed to *they can them*, which is grammatical with the transitive verb use of *can*).

```
VP-pl -> V-pl NP-sg
VP-pl -> V-pl NP-pl
NP-sg -> he
NP-sg -> fish
NP-pl -> fish
```

Note that we have to expand out the symbols even when there's no constraint on agreement, since we have no way of saying that we don't care about the value of number for a category (e.g., past tense verbs).

Another linguistic phenomenon that we are failing to deal with is *subcategorization*. This is the lexical property that tells us how many *argument*s a verb can have (among other things). Subcategorization tends to mirror semantics, although there are many complications. A verb such as *adore*, for instance, relates two entities and is transitive: a sentence such as **Kim adored* is strange, while *Kim adored Sandy* is usual. A verb such as *give* is *ditransitive*: *Kim gave Sandy an apple* (or *Kim gave an apple to Sandy*). Without going into details of exactly how subcategorization is defined, or what an argument is, it should be intuitively obvious that we're not encoding this property with our CFG. The grammar in lecture 4 allows the following, for instance:

> *they fish fish it*
> (S (NP they) (VP (V fish) (VP (V fish) (NP it))))

Again this could be dealt with by multiplying out symbols (V-intrans, V-ditrans etc), but the grammar becomes extremely cumbersome.

Finally, consider the phenomenon of *long-distance dependencies*, exemplified, for instance, by:

> which problem did you say you don't understand?
> who do you think Kim asked Sandy to hit?
> which kids did you say were making all that noise?

Traditionally, each of these sentences is said to contain a *gap*, corresponding to the place where the noun phrase would normally appear: the gaps are marked by underscores below:

> which problem did you say you don't understand _?
> who do you think Kim asked Sandy to hit _?
> which kids did you say _ were making all that noise?

Notice that, in the third example, the verb *were* shows plural agreement.

Doing this in standard CFGs is possible, but extremely verbose, potentially leading to trillions of rules. Instead of having simple atomic categories in the CFG, we want to allow for features on the categories, which can have values indicating things like plurality. As the long-distance dependency examples should indicate, the features need to be complex-valued. For instance,

> * what kid did you say _ were making all that noise?

is not grammatical. The analysis needs to be able to represent the information that the gap corresponds to a plural noun phrase.

4.12 Further reading

This lecture has described material which J&M discuss in chapters 12 and 13, though we also touched on PCFGs (covered in their chapter 14) and issues of language complexity which they discuss in chapter 16. I chose to concentrate on bottom-up chart parsing in this lecture, mainly because I find it easier to describe than the Earley algorithm and the full version of chart parsing given in J&M, but also because it is easier to see how to extend this to PCFGs.

Bender 2013 (listed in the introduction) has a succinct discussion of constituency and other syntactic issues alluded to here.

A glossary/index of some of the terms used in the lectures

This is primarily intended to cover concepts which are mentioned in more than one lecture. The lecture where the term is explained in most detail is generally indicated. In some cases, I have just given a pointer to the section in the lectures where the term is defined. Note that IGE stands for *The Internet Grammar of English* (`http://www.ucl.ac.uk/internet-grammar/home.htm`). There are a few cases where this uses a term in a slightly different way from these course notes: I have tried to indicate these.

adjective See IGE or notes for pre-lecture exercises in lecture 3.

adjunct See **argument** and also IGE.

adverb See IGE or notes for pre-lecture exercises in lecture 3.

affix A morpheme which can only occur in conjunction with other morphemes (lecture 2).

AI-complete A half-joking term, applied to problems that would require a solution to the problem of representing the world and acquiring world knowledge (lecture 1).

agreement The requirement for two phrases to have compatible values for grammatical features such as number and gender. For instance, in English, *dogs bark* is grammatical but *dog bark* and *dogs barks* are not. See IGE.

ambiguity The same string (or sequence of sounds) meaning different things. Contrasted with **vagueness**.

anaphora The phenomenon of referring to something that was mentioned previously in a text. An anaphor is an expression which does this, such as a pronoun (see lecture 10).

antonymy Opposite meaning: such as *clean* and *dirty* (lecture 7).

argument In syntax, the phrases which are lexically required to be present by a particular word (prototypically a verb). This is as opposed to **adjunct**s, which modify a word or phrase but are not required. For instance, in:

> Kim saw Sandy on Tuesday

Sandy is an argument but *on Tuesday* is an adjunct. Arguments are specified by the **subcategorization** of a verb etc. Also see the IGE.

aspect A term used to cover distinctions such as whether a verb suggests an event has been completed or not (as opposed to tense, which refers to the time of an event). For instance, *she was writing a book* vs *she wrote a book*.

backoff Usually used to refer to techniques for dealing with data sparseness in probabilistic systems: using a more general classification rather than a more specific one. For instance, using unigram probabilities instead of bigrams; using word classes instead of individual words (lecture 3).

bag of words Unordered collection of words in some text.

baseline In evaluation, the performance produced by a simple system against which the experimental technique is compared (§3.6).

bidirectional Usable for both analysis and generation (lecture 2).

case Distinctions between nominals indicating their syntactic role in a sentence. In English, some pronouns show a distinction: e.g., *she* is used for subjects, while *her* is used for objects. e.g., *she likes her* vs **her likes she*. Languages such as German and Latin mark case much more extensively.

ceiling In evaluation, the performance produced by a 'perfect' system (such as human annotation) against which the experimental technique is compared (§3.6).

CFG context-free grammar.

chart parsing See §4.5.

Chomsky Noam Chomsky, professor at MIT. His work underlies most modern approaches to syntax in linguistics. Not so hot on probability theory.

classifier A system which assigns classes to items, usually using a machine learning approach.

closed class Refers to parts of speech, such as conjunction, for which all the members could potentially be enumerated (lecture 3).

collocation See lecture 7.

complement For the purposes of this course, an **argument** other than the subject.

compositionality The idea that the meaning of a phrase is a function of the meaning of its parts. **compositional semantics** is the study of how meaning can be built up by semantic rules which mirror syntactic structure (lecture 6).

constituent A sequence of words which is considered as a unit in a particular grammar (lecture 4).

constraint-based grammar A formalism which describes a language using a set of independently stated constraints, without imposing any conditions on processing or processing order.

context The situation in which an utterance occurs: includes prior utterances, the physical environment, background knowledge of the speaker and hearer(s), etc etc. Nothing to do with context-free grammar.

corpus A body of text used in experiments (plural *corpora*). See §3.1.

cue phrases Phrases which indicates particular **rhetorical relations**.

denominal Something derived from a noun: e.g., the verb *tango* is a denominal verb.

dependency structure A syntactic or semantic representation that links words via relations See Lecture 5.

derivational morphology See §2.2

determiner See IGE or notes for pre-lecture exercises in lecture 3.

deverbal Something derived from a verb: e.g., the adjective *surprised*.

direct object See IGE. Contrast **indirect object**.

distributional semantics Representing word meaning by context of use (lecture 8).

discourse In NLP, a piece of connected text.

discourse relations See **rhetorical relations**.

domain Not a precise term, but I use it to mean some restricted set of knowledge appropriate for an application.

error analysis In evaluation, working out what sort of errors are found for a given approach (§3.6).

expletive pronoun Another term for **pleonastic pronoun**: see Lecture 10.

feature A characteristic property used in machine learning.

FSA Finite state automaton

FST Finite state transducer

full-form lexicon A lexicon where all morphological variants are explicitly listed (lecture 2).

generation The process of constructing text (or speech) from some input representation (lecture 11).

generative grammar The family of approaches to linguistics where a natural language is treated as governed by rules which can produce all and only the well-formed utterances. Lecture 4.

genre Type of text: e.g., newspaper, novel, textbook, lecture notes, scientific paper. Note the difference to **domain** (which is about the type of knowledge): it's possible to have texts in different genre discussing the same domain (e.g., discussion of human genome in newspaper vs textbook vs paper).

gloss An explanation/translation of an obscure/foreign word or phrase.

grammar Formally, in the generative tradition, the set of rules and the lexicon. Lecture 4.

head In syntax, the most important element of a phrase.

hearer Anyone on the receiving end of an utterance (spoken, written or signed). §1.3.

Hidden Markov Model See §3.5

HMM Hidden Markov Model

homonymy Instances of **polysemy** where the two senses are unrelated (Lecture 7).

hyponymy An 'IS-A' relationship (Lecture 7) More general terms are **hypernym**s, more specific **hyponym**s.

indirect object The beneficiary in verb phrases like *give a present to Sandy* or *give Sandy a present*. In this case the indirect object is *Sandy* and the **direct object** is *a present*.

interannotator agreement The degree of agreement between the decisions of two or more humans with respect to some categorisation (§3.6).

language model A term generally used in speech recognition, for a statistical model of a natural language (lecture 3).

lemmatization Finding the stem and affixes for words (lecture 2).

lexical ambiguity Ambiguity caused because of multiple senses for a word.

lexicon The part of an NLP system that contains information about individual words (lecture 1).

local ambiguity Ambiguity that arises during analysis etc, but which will be resolved when the utterance is completely processed.

logical form The semantic representation constructed for an utterance (lecture 6).

long-distance dependency See §4.11

meaning postulates Inference rules that capture some aspects of the meaning of a word.

meronymy The 'part-of' lexical semantic relation (Lecture 7).

modifier Something that further specifies a particular entity or event: e.g., *big house*, *shout loudly*.

morpheme Minimal information carrying units within a word (§2.1).

morphology See §1.2

multiword expression A fixed phrase with a non-compositional meaning (lecture 6).

MT Machine translation.

multiword expression A conventional phrase that has something idiosyncratic about it and therefore might be listed in a dictionary.

mumble input Any unrecognised input in a spoken dialogue system (lecture 2).

n-gram A sequence of n words (§3.2).

named entity recognition Recognition and categorisation of person names, names of places, dates etc (lecture 4).

NL Natural language.

NLG Natural language generation (lecture 11).

NLID Natural language interface to a database.

nominal In grammar terminology, noun-like (can be used to describe a word or a phrase).

noun See IGE or notes for pre-lecture exercises in lecture 3.

noun phrase (NP) A phrase which has a noun as syntactic **head**. See IGE.

ontology In NLP and AI, a specification of the entities in a particular domain and (sometimes) the relationships between them. Often hierarchically structured.

open class Opposite of **closed class**.

orthographic rules Same as **spelling rules** (§2.4)

overgenerate Of a grammar, to produce strings which are invalid, e.g., because they are not grammatical according to human judgements.

packing See §4.8

passive chart parsing See §4.6

parse tree See §4.4

part of speech The main syntactic categories: noun, verb, adjective, adverb, preposition, conjunction etc.

part of speech tagging Automatic assignment of syntactic categories to the words in a text. The set of categories used is actually generally more fine-grained than traditional parts of speech.

pleonastic Non-referring (esp. of pronouns): see Lecture 10.

polysemy The phenomenon of words having different senses (Lecture 7).

POS Part of speech (in the context of POS tagging).

pragmatics See §1.2

predicate In logic, something that takes zero or more arguments and returns a truth value. (Used in IGE for the verb phrase following the subject in a sentence, but I don't use that terminology.)

prefix An **affix** that precedes the **stem**.

probabilistic context free grammars (PCFGs) CFGs with probabilities associated with rules (lecture 4).

realization Construction of a string from a meaning representation for a sentence or a syntax tree (lecture 11).

referring expression See Lecture 10

relative clause See IGE.
A **restrictive relative clause** is one which limits the interpretation of a noun to a subset: e.g. *the students who sleep in lectures are obviously overworking* refers to a subset of students. Contrast **non-restrictive**, which is a form of parenthetical comment: e.g. *the students, who sleep in lectures, are obviously overworking* means all (or nearly all) are sleeping.

selectional restrictions Constraints on the semantic classes of arguments to verbs etc (e.g., the subject of *think* is restricted to being sentient). The term **selectional preference** is used for non-absolute restrictions.

semantics See §1.2

smoothing Redistributing observed probabilities to allow for **sparse data**, especially to give a non-zero probability to unseen events (lecture 2).

SMT Statistical machine translation.

sparse data Especially in statistical techniques, data concerning rare events which isn't adequate to give good probability estimates (lecture 2).

speaker Someone who makes an **utterance** (§1.3).

spelling rules §2.4

stem A **morpheme** which is a central component of a word (contrast **affix**). §2.1.

stemming Stripping **affix**es (see §2.3).

strong equivalence Of grammars, accepting/rejecting exactly the same strings and assigning the same bracketings (contrast **weak equivalence**). Lecture 4.

structural ambiguity The situation where the same string corresponds to multiple bracketings.

subcategorization The lexical property that tells us how many **argument**s a verb etc can have.

suffix An **affix** that follows the **stem**.

summarization Producing a shorter piece of text (or speech) that captures the essential information in the original.

synonymy Having the same meaning (Lecture 7).

syntax See §1.2

taxonomy Traditionally, the scheme of classification of biological organisms. Extended in NLP to mean a hierarchical classification of word senses. The term **ontology** is sometimes used in a rather similar way, but ontologies tend to be classifications of domain-knowledge, without necessarily having a direct link to words, and may have a richer structure than a taxonomy.

tense Past, present, future etc.

training data Data used to train any sort of machine-learning system. Must be separated from test data which is kept unseen. Manually-constructed systems should also use strictly unseen data for evaluation.

treebank a corpus annotated with trees (lecture 4).

weak equivalence Of grammars, accepting/rejecting exactly the same strings (contrast **strong equivalence**). Lecture 4.

Wizard of Oz experiment An experiment where data is collected, generally for a dialogue system, by asking users to interact with a mock-up of a real system, where some or all of the 'processing' is actually being done by a human rather than automatically.

WordNet See Lecture 7

WSD Word-sense disambiguation

utterance A piece of speech or text (sentence or fragment) generated by a speaker in a particular context.

vagueness Of word meanings, contrasted with **ambiguity** : see Lecture 7.

verb See IGE or notes for pre-lecture exercises in lecture 3.

verb phrase (VP) A phrase headed by a verb.

Exercises for NLP course, lecture 1–4, 2017

Notes on exercises

These exercises are organised by lecture. They are divided into two classes: pre-lecture and post-lecture. The pre-lecture exercises are intended to review the basic concepts that you'll need to fully understand the lecture. Depending on your background, you may find these trivial or you may need to read the notes, but in either case they shouldn't take more than a few minutes. The first one or two examples generally come with answers, other answers are at the end (where appropriate).

Answers to the post-lecture exercises are available to supervisors (where appropriate). These are mostly intended as quick exercises to check understanding of the lecture, though some are more open-ended.

A Lecture 1

A.1 Post-lecture exercises

Without looking at any film reviews beforehand, write down 10 words which you think would be good indications of a positive review (when taken in isolation) and 10 words which you think would be negative. Then go through a review of a film and see whether you find there are more of your positive words than the negative ones. Are there words in the review which you think you should have added to your initial lists?

Have a look at `http://www.cl.cam.ac.uk/~aac10/stuff.html` for pointers to sentiment analysis data used in experiments.

B Lecture 2

B.1 Pre-lecture exercises

1. Split the following words into morphological units, labelling each as stem, suffix or prefix. If there is any ambiguity, give all possible splits.

   (a) dries
   answer: dry (stem), -s (suffix)

   (b) cartwheel
   answer: cart (stem), wheel (stem)

   (c) carries

   (d) running

   (e) uncaring

   (f) intruders

   (g) bookshelves

   (h) reattaches

   (i) anticipated

2. List the simple past and past/passive participle forms of the following verbs:

   (a) sing
   Answer: simple past *sang*, participle *sung*

   (b) carry

   (c) sleep

(d) see

Note that the simple past is used by itself (e.g., *Kim sang well*) while the participle form is used with an auxiliary (e.g., *Kim had sung well*). The passive participle is always the same as the past participle in English: (e.g., *Kim began the lecture early*, *Kim had begun the lecture early*, *The lecture was begun early*).

B.2 Post-lecture exercises

1. For each of the following surface forms, give a list of the states that the FST given in the lecture notes for e-insertion passes through, and the corresponding underlying forms:

   (a) c a t s

   (b) c o r p u s

   (c) a s s e s

   (d) a s s e s s

   (e) a x e s

2. Modify the FSA for dates so that it only accepts valid months. Turn your revised FSA into a FST which maps between the numerical representation of months and their abbreviations (Jan . . . Dec).

3. Earlier dialects of English used expressions like "four and twenty" instead of "twenty four" (the same pattern occurs in modern German, French has something similar but mostly omits the 'and'). This pattern applies for numbers from 21 to 99, with the exception of 30, 40 etc. Assume a numerical version of this, where '4&20' corresponds to '24', '3&30' to '33' etc. Write a FST that represents this mapping for numbers from 1 to 99 ('14', '4' etc should correspond to themselves). What does this illustrate about the FST formalism?

4. The lecture talks about morphological paradigms: suggest appropriate slots for English verbs (you may ignore *be*) and give an example of a regular and irregular verb using this paradigm.

5. Look up the rules for the original Porter stemmer and give 5 examples of words where these could give significantly different results on a task like sentiment analysis compared to finding the linguistic stem.

6. Outline how you might write a FSA corresponding to dialogue states for booking a table in a known restaurant (the full FSA is not required).

C Lecture 3

C.1 Pre-lecture

Label each of the words in the following sentences with their part of speech, distinguishing between nouns, proper nouns, verbs, adjectives, adverbs, determiners, prepositions, pronouns and others. (Traditional classifications often distinguish between a large number of additional parts of speech, but the finer distinctions won't be important here.) There are notes on part of speech distinctions below, if you have problems.

1. The brown fox could jump quickly over the dog, Rover. Answer: The/Det brown/Adj fox/Noun could/Verb(modal) jump/Verb quickly/Adverb over/Preposition the/Det dog/Noun, Rover/Proper noun.

2. The big cat chased the small dog into the barn.

3. Those barns have red roofs.

4. Dogs often bark loudly.

5. Further discussion seems useless.

6. Kim did not like him.

7. Time flies.

Notes on parts of speech. These notes are English-specific and are just intended to help with the lectures and the exercises: see a linguistics textbook for definitions! Some categories have fuzzy boundaries, but none of the complicated cases will be important for this course.

Noun prototypically, nouns refer to physical objects or substances: e.g., *aardvark*, *chainsaw*, *rice*. But they can also be abstract (e.g. *truth*, *beauty*) or refer to events, states or processes (e.g., *decision*). If you can say *the X* and have a sensible phrase, that's a good indication that X is a noun.

Pronoun something that can stand in for a noun: e.g., *him*, *his*

Proper noun / Proper name a name of a person, place etc: e.g., *Elizabeth*, *Paris*

Verb Verbs refer to events, processes or states but since nouns and adjectives can do this as well, the distinction between the categories is based on distribution, not semantics. For instance, nouns can occur with determiners like *the* (e.g., *the decision*) whereas verbs can't (e.g., * *the decide*). In English, verbs are often found with auxiliaries (*be*, *have* or *do*) indicating tense and aspect, and sometime occur with modals, like *can*, *could* etc. Auxiliaries and modals are themselves generally treated as subclasses of verbs.

Adjective a word that modifies a noun: e.g., *big*, *loud*. Most adjectives can also occur after the verb *be* and a few other verbs: e.g., *the students are unhappy*. Numbers are sometimes treated as a type of adjective by linguists but generally given their own category in traditional grammars. Past participle forms of verbs can also often be used as adjectives (e.g., *worried* in *the very worried man*). Sometimes it's impossible to tell whether something is a participle or an adjective (e.g., *the man was worried*).

Adverb a word that modifies a verb: e.g. *quickly*, *probably*.

Determiner these precede nouns e.g., *the*, *every*, *this*. It is not always clear whether a word is a determiner or some type of adjective.

Preposition e.g., *in*, *at*, *with*

Nouns, proper nouns, verbs, adjectives and adverbs are the *open classes*: new words can occur in any of these categories. Determiners, prepositions and pronouns are closed classes (as are auxiliary and modal verbs).

C.2 Post-lecture

1. Try out one or more of the POS tagging sites linked from:
   `http://www.cl.cam.ac.uk/~aac10/stuff.html`

   Find two short pieces of naturally occurring English text, one of which you think should be relatively easy to tag correctly and one which you predict to be difficult. Look at the tagged output and estimate the percentage of correct tags in each case, concentrating on the open-class words. You might like to get another student to look at the same output and see if you agree on which tags are correct.

2. (OPEN-ENDED) The notes briefly discuss word prediction for use in an AAC system. Describe how such an approach might be enhanced by the use of POS tags. What type of issues might you expect to arise in developing such a system?

3. The notes mention that verbal uses of words that are mainly nouns (e.g., *tango*) are quite likely to be mistagged. Investigate this with an online part-of-speech tagger and try and explain your results on the basis of the HMM model discussed in the lectures (though not all taggers use HMMs).

D Lecture 4

D.1 Pre-lecture

Put brackets round the noun phrases and the verb phrases in the following sentences (if there is ambiguity, give two bracketings):

1. The cat with white fur chased the small dog into the barn.
   Answer: ((The cat)$_{np}$ with (white fur)$_{np}$)$_{np}$ chased (the small dog)$_{np}$ into (the barn)$_{np}$
   The cat with white fur (chased the small dog into the barn)$_{vp}$

2. The big cat with black fur chased the dog which barked.

3. Three dogs barked at him.

4. Kim saw the birdwatcher with the binoculars.

Note that noun phrases consist of the noun, the determiner (if present) and any modifiers of the noun (adjective, prepositional phrase, relative clause). This means that noun phrases may be nested. Verb phrases include the verb and any auxiliaries, plus the object and indirect object etc (in general, the complements of the verb) and any adverbial modifiers.[29] The verb phrase does not include the subject.

D.2 Post-lecture

1. Using the CFG given in the lecture notes (section 4.3):

   (a) show the edges generated when parsing *they fish in rivers in December* with the simple chart parser in 4.7

   (b) show the edges generated for this sentence if packing is used (as described in 4.9)

2. Modify the CFG given in section 4.3 so that intransitive, transitive and ditransitive verbs are distinguished (add suitable lexical entries to illustrate this).

3. Modify the CFG to allow for adverbs.

4. Suppose the following rules are added to the grammar:

   ```
   VP -> VP Adv
   S -> S Adv
   ```

   What would this imply for the analysis of a sentence like: *Kim barked loudly*? If only the first rule were present, would the resulting grammar be weakly- or strongly- equivalent to the original? Could there be reasons to want both rules?

E Answers to some of the exercises

E.1 Lecture 1 (post-lecture)

Something like this experiment was tried by Pang et al (2002) to provide a baseline for their machine learning system. The table below shows the accuracy they obtained on movie reviews by counting the positive and negative terms in the document. The third set was obtained with the help of preliminary frequency data: note the inclusion of '?' and '!'.

[29] A *modifier* is something that further specifies a particular entity or event: e.g., *big house*, *shout loudly*.

	Terms	Accuracy
Human 1	positive: *dazzling, brilliant, phenomenal, excellent, fantastic* negative: *suck, terrible, awful, unwatchable, hideous*	58%
Human 2	positive: *gripping, mesmerizing, riveting, spectacular, cool, awesome, thrilling, badass, excellent, moving, exciting* negative: *bad, cliched, sucks, boring, stupid, slow*	64%
Human 3 (with stats)	positive: *love, wonderful, best, great, superb, still, beautiful* negative: *bad, worst, stupid, waste, boring, ?, !*	69%

E.2 Lecture 2 (pre-lecture)

1. (a) carries
   carry (stem) s (suffix)
   (b) running
   run (stem) ing (suffix)
   (c) uncaring
   un (prefix) care (stem) ing (suffix)
   (d) intruders
   intrude (stem) er (suffix) s (suffix)
   Note that in- is not a real prefix here
   (e) bookshelves
   book (stem) shelf (stem) s (suffix)
   (f) reattaches
   re (prefix) attach (stem) s (suffix)
   (g) anticipated
   anticipate (stem) ed (suffix)
2. (a) carry
   Answer: simple past *carried*, past participle *carried*
   (b) sleep
   Answer: simple past *slept*, past participle *slept*
   (c) see
   Answer: simple past *saw*, past participle *seen*

E.3 Lecture 3 (pre-lecture)

1. The/Det big/Adj cat/Noun chased/Verb the/Det small/Adj dog/Noun into/Prep the/Det barn/Noun.
2. Those/Det barns/Noun have/Verb red/Adj roofs/Noun.
3. Dogs/Noun often/Adverb bark/Verb loudly/Adverb.
4. Further/Adj discussion/Noun seems/Verb useless/Adj.
5. Kim/Proper noun did/Verb(aux) not/Adverb(or Other) like/Verb him/Pronoun.
6. Time/Noun flies/Verb.
   Time/Verb flies/Noun. (the imperative!)

E.4 Lecture 4 (pre-lecture)

1. The big cat with black fur chased the dog which barked.
   ((The big cat)$_{np}$ with (black fur)$_{np}$)$_{np}$ chased (the dog which barked)$_{np}$
   The big cat with black fur (chased the dog which barked)$_{vp}$

2. Three dogs barked at him. (Three dogs)$_{np}$ barked at (him)$_{np}$ Three dogs (barked at him)$_{vp}$

3. Kim saw the birdwatcher with the binoculars.
   Analysis 1 (the birdwatcher has the binoculars) (Kim)$_{np}$ saw ((the birdwatcher)$_{np}$ with (the binoculars)$_{np}$)$_{np}$
   Kim (saw the birdwatcher with the binoculars)$_{vp}$
   Analysis 2 (the seeing was with the binoculars) (Kim)$_{np}$ saw (the birdwatcher)$_{np}$ with (the binoculars)$_{np}$
   Kim (saw the birdwatcher with the binoculars)$_{vp}$