

CS 662 Notes

Jonathan May

August 23, 2021

Chapter 1

Introduction

1.1 What is NLP?

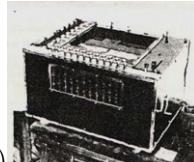
- Analysis of a natural language (wait, what's a natural language)
- Generation of a natural language
- Sometimes both analysis and generation
- Representation of a natural language (but usually in the service of generation and/or analysis)

1.1.1 Super Brief History of NLP

- NLP was indistinguishable from MT for a long time for some weird reason
- Athanasius Kircher – ‘polygraphy’ (1630) for translation without understanding
 - side note: kircher is really weird guy! he
 - incorrectly deciphered hieroglyphics
 - Went into Vesuvius’ erupting crater for his geology research
 - invented a cat piano
 - check him out at the Museum of Jurassic Technology just outside Culver City
- Becher mechanical meta-language for language-to-meaning (in the service of NLP):



1666



- Georges Artsruni mechanical brain (1930)
- Computers proposed for translation (1949...3 years after computer invented)



- ACL founded (as the “Association for Machine Translation”) in 1962

1.1.2 Applications

- Chatbots
- customer service agents
- Question Answering (search)
- Translation
- Grammar Correction
- Summarization
- Article Writing
- Opinion/sentiment analysis
- essay evaluation/plagiarism detection
- Knowledge base construction
- author identification
- ...

1.1.3 A Dialogue of the near-future

HUMAN: Can you get me a table at N/Naka tomorrow at 7?

AI: How many?

H: Four of us.

A: Sorry, there's nothing at 7 PM. They have an opening at 10 PM.

H: Ugh, that's sort of late. What about Sakura?

A: Sakura has an opening at 9:30 PM.

H: Can you email Brooke, Mike, and Liz, and ask if 9:30 is too late?

A: OK, email sent.

A: Liz says it's fine.

H: OK, let's book the table at Sakura.
A: OK, all set for 4 people at 9:30 PM.
[some time later...]
H: You know what, just take the N/Naka one at 10 PM.
A: OK, N/Naka is booked at 10 PM for 4 people and I cancelled the reservation at Sakura.

What parts of this conversation are possible with NLP tech today (cf. Google Duplex demo)? Which aren't? Which seem likely to be coming soon, based on your intuition?

1.1.4 Find the following NL ‘tasks’ in the above dialogue, or see where they would be useful

- Spelling Correction
- Named Entity Information Extraction
- Question Answering
- Coreference Resolution
- Grounding
- Ambiguity Resolution
- Summarization
- Information Retrieval
- Generation
- Translation

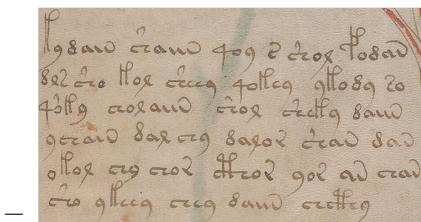
1.1.5 Connections to Other Areas

- Machine Learning – the biggie, now. NLP: ML is a tool we use. ML: NLP is a data set we use.
- Linguistics – for ML folks think of this as ‘guided search’ both within a problem and when considering what the problems are. But also consider we are trying to understand language and can use NLP/CL techniques to do so
- Cognitive Science / Psychology – see also Linguistics, but a level up. We are fascinated by humans’ ability to learn language that is far better than computers’ and we’re not quite sure why this is. We probably won’t get into real cog sci/psych theories, though
- information theory – language is a means by which humans communicate and the communication capacity/compression/confusability of communications is baked in to our studies, particularly when discussing (cross-)entropy and mutual information

- theory of computation – very important for search over complex spaces (e.g. for a syntactic tree or semantic graph) and for recognizing what transformations are and are not possible. Strongly connected via chomsky hierarchy, of which we'll only talk a little
- data science, political science, etc. – NLP: these are good subdomains to try our stuff. X science: NLP is a good tool to demonstrate my theories
- Other areas you're interested in not covered here?

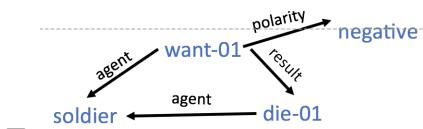
1.2 Linguistic Stack – from low to high ambiguity (from shallow to deep)

- pre-text (speech channel)
 - phonetics – what mouth sound has been produced?
 - * [l] = alveolar lateral approximant (lace)
 - * [r] = alveolar tap (race)
 - * [ɾ] = alveolar trill (rey (Spanish))
 - phonology – what are the meaningfully distinct sounds (governed by each language)?
 - * English: [f] vs. [r] conflated
 - * Japanese: [r] vs. [ɾ] vs. [l] conflated
 - * Hindi: ḍ (dental) vs. ḍ̥ (dental, glottal) distinct, etc.
 - * cf https://en.wikipedia.org/wiki/Hindustani_phonology examples (retroflex are good ones to confuse my american ears)
- pre-text (vision channel)
 - orthography – what makes a character? Particularly difficult when dealing with unknown writing set, especially handwritten.



- morphology – what are the minimal meaning-laden parts of a word that are useful to distinguish? (Why distinguish? For handling novelty (wug test), collapsing statistics...)
- English is pretty weak here:
- inflection: ‘talks’ = ‘talk (verb) + s (present 3rd singular)’ or ‘talk (noun) + s (plural).’

- Turkish agglutination: *uygarlaştıramadıklarımızdanmissınızcasına* = “(behaving) as if you are among those whom we could not civilize”
- words (lexemes)
 - Ok, not really a level but it’s important to recognize what we consider a word, especially when data processing
 - Is a word ‘a single unit of meaning?’ ‘text separated by whitespace?’
 - What about Chinese? Or Thai? Or long compounds/agglutinations in Turkish, German, Finnish?
 - What about whitespace-separated units that function noncompositionally (‘New York’, ‘take out’)?
 - What about hyphenated and punctuated text? (Tricky example: ‘New York-New Haven Railroad’)
- syntax – how to properly put words together to form a sentence
 - part-of-speech tags:
 - the/DT (determiner) blue/JJ (adjective) boat/NN (noun)
 - *boat/NN blue/JJ the/DT
 - Constituencies:
 - S = NP[the blue boat] VP[sailed home]
 - * VP[sailed home] NP[the blue boat]
- semantics – what does a word in a sentence mean, and how do the words meaningfully relate to each other?
 - Consider the sentence ‘The soldier did not want to die.’
 - What is meant by ‘want’ – desire? lack?
 - Who is doing the wanting? Who is doing the dying? What is (not) wanted?



- pragmatics – what does the *speaker* (as opposed to the sentence) mean in conversational context?

HUMAN: Can you get me a table at N/Naka tomorrow at 7?

*AI: Yes, I have that ability.

AI: OK, your reservation is made.

- discourse – what information is conveyed subtextually, as a result of context (interpretation of sentence in context to other sentence or sequence, overall intent of text or dialogue)?

WAITER: What would you like for dinner?

DINER: I had a heavy lunch.

WAITER: Let me tell you about our salads.

1.3 Ambiguity is the enemy of NLP

Humans are seemingly able to integrate lots of context, world knowledge, tone, etc. clues to clearly disambiguate ‘bank’ and ‘mean’ and ‘latex’, find no problem in deciding to not pick apart individual meaning in ‘make a decision’ or ‘take out’ or ‘make up’, and can easily conclude that if you can *gronfle* sixty *milchanks* in one hour, then after one minute you will have *gronfled* one *milchank*. Errors in NLP are chiefly due to not having sufficient context.

1.3.1 Funny (English) Examples made less funny by linguistic analysis – pick out the misinterpreted phenomenon!

- Enraged Cow Injures Farmer With Axe
- Ban on Nude Dancing on Governor’s Desk
- Teacher Strikes Idle Kids
- Hospitals are Sued by 7 Foot Doctors
- Iraqi Head Seeks Arms
- Stolen Painting Found by Tree
- Kids Make Nutritious Snacks
- Local HS Dropouts Cut in Half
- Dinosaurs didn’t read. Now they are extinct.

1.3.2 Issues in ambiguity, richness

“We saw the woman with the telescope wrapped in paper” – who has the telescope? What is the paper wrapping? Is this perception or assault? Humans have two major readings of this (why?) but it’s hard to keep computers from considering the unlikely ones unlikely.

“Every fifteen minutes a woman in this country gives birth. Our job is to find that woman, and stop her!” Groucho Marx – ambiguity of semantics (‘a woman’)

‘The soldier was not afraid to die’ vs ‘The soldier did not fear death.’ – There are lots of ways to express the same thing and to us this is not an issue, but without proper intervention these are completely distinct sentences to a computer model.

Humans often produce intentionally obfuscated language, possibly to only target a subgroup. These obfuscations can change very quickly; it's tough to keep up! There are good reasons to keep up! Examples:

- Call me at six niner i three triple 0 dos
- u ship them? smh. ikr, lolol, yolo.
- 14 words now and then 88 later if u want

1.4 Structure Of the class

- The overall structure of learning is:
 - a tiny bit of linguistics
 - a refresher on probability you should know
 - mini-course on important aspects of machine learning (linear models, nonlinear models)
 - discussion of data
 - discussion of evaluation
 - some core techniques oriented toward parts of the linguistic stack
 - various end-goal subfields (MT, IE, Dialogue, maybe QA)
- I also want you to get experienced digesting the latest papers and to break the class up a bit so I'm not lecturing the whole time. Everybody will choose one paper from NAACL 2021 <https://aclanthology.org/events/naacl-2021/> and will make a 15-minute presentation on it in class. (You can't present your own work, BTW.) We'll all read the paper ahead of time and engage in active discussion on the paper using piazza and in-class.
- Based on class interest we can add or subtract topics, especially near the end of the class.
- There are three HW assignments and a project. As shown below, the majority of your effort should be in writing clearly and communicating what you have done.

1.5 Evaluation

- no punishment curve (but a reward curve if needed);
- be an active, engaged participant and do all the work and an A is easy to get.

1.5.1 Homeworks – 3x10% = 30% total

- Expect substantial programming in most (if not all) of them
- Coding should be strictly in Python – in some cases we may provide useful templates for parts of the assignment.
- Writeups should be strictly in L^AT_EX; learn how to use it now (if you don't already)!
- Communicate well; write clearly and simply, use appropriate figures and graphs.
- There will be an expectation of self-exploration, reading papers to get good ideas to reimplement or build upon.
- **ALL CODE MUST BE YOUR OWN AND MAY NOT BE COPIED** which includes solutions you find on the web and code from others in the class. We run code-checking software; it is smart enough to defeat attempts to obfuscate, and I take cheating very seriously (see below).
- Grading (approximate and totally subjective) presuming that you actually did what was asked:
 - about 50% – did you clearly communicate your description of what you implemented, how you implemented it, what your experiments were, and what conclusions you drew from them? This includes appropriate use of graphics and tables where warranted that clearly explain your point. This also includes well written explanations that tell a compelling story. Grammar and syntax are a small part of this (maybe 5%) but much more important is the narrative you tell. Also a part of this is that you clearly acknowledged your sources and influences with appropriate bibliography and, where relevant, cited influencing prior work.
 - about 20% – is your code correct? Did you implement what was asked for, and did you do it correctly?
 - about 20% – is your code well-written, documented, and robust? Will it run from a different directory than the one you ran it in? Does it rely on hard-codes? Is it commented and structured such that we can read it and understand what you are doing?
 - about 10% – did you go the extra mile? Did you push beyond what was asked for in the assignment, trying new models, features, or approaches? Did you use motivation (and document appropriately) from another researcher trying the same problem or from an unrelated but transferrable other paper? **THIS IS NOT EXTRA CREDIT! YOU CANNOT RECEIVE 100% WITHOUT COMPLETING THIS PART!**. There is no extra credit on homeworks.

1.5.2 In-class paper presentation = 10%

- The schedule contains paper listings under “presentation”; everyone will sign up for (at least one) NAACL 2021 paper to present to the class and lead discussion (approx 15 minutes presentation, 5-10 minutes discussion, but this is flexible)

- Everyone will read the papers (see below) and prepare questions ahead of time to facilitate discussion
- You will explain the paper, getting into key details and insights as well as the context of the paper (you may have to look at key papers that cited this paper as well as key influential works that the paper cites)
- Slides or a handout may be helpful and are a good idea but are not mandatory
- Submit your top 3 papers as a response to a poll we will send; we'll do our best to give everyone their top choice but can't promise. If you don't pick one we'll pick for you!

1.5.3 Project—5% (proposal) + 5% (version 1 of report) + 10% (final presentation) + 20%(final report) = 40%

- Two people per project
- Reproduce results in an ACL 2021 paper
- Write up your results clearly
- Proposal is due in two weeks from the beginning of class!

1.5.4 Pre-written paper presentation questions for others' presentations— 10/(N-1)% x N-1 = 10%

- Before the class in which a presentation is going to be given (i.e. in 2020, by 9:59 am Pacific time on the day of the class), post at least one question regarding each presentation (usually 1 but occasionally 2) to the appropriate location on piazza.
- During the presentation, if the question isn't answered or isn't answered sufficiently, bring it up and engage in discussion with the class.
- There probably won't be enough time for everyone; if the question is unresolved it should be discussed on piazza (after or before the presentation)
- The main goal of this is to ensure that you've read the papers and are engaging in discussion.

1.5.5 Other class participation—10%

- Ask general questions in class, engage in discussion
- Ask questions and engage in discussion on piazza – answer each others' questions before instructors weigh in
- Propose topics to cover

- Ask questions of fellow students during project presentations
- Answer questions when I call on you and be in class (occasional absences understandable).

1.5.6 Late Days

- Work is done at 11:59:59 anywhere on earth (=4 AM the next day, for PST) on the announced due date (except for the final report).
- You get four *cumulative* late days for homeworks and project proposals (no late days for final project report or missed presentations). Thereafter, 20% off per day.
- Late group project proposals will deduct from both team members' late day accounts.

1.5.7 Office Hours

- 2021: Mine are 2pm–3pm before class, in SAL 311 (or on zoom).
- Elan's are TBD
- Come bounce ideas off of us (particularly related to project proposal/project)

1.6 Don't Cheat

- Read the USC honor code; it applies, and I will abide by it.
- All work you turn in should be your own.
- This includes anything written and all code.
- All work must be originally done **for this class** by you. Self-plagiarizing is still plagiarizing!
- That means that if you have done any of these assignments before in a previous class, you should either not look at your previous work, or (better) come talk to me and I will give you an alternate assignment
- Similarly, for the project, you may not re-present a research project or paper you have previously worked on or are currently working on; this should be entirely new work.
- If we have determined you have violated the honor code we will invoke punishments as deemed necessary; this can mean a zero on an assignment, a reduced letter grade in the class, or even a failing grade. Punishments can occur at any time after violations (usually at the worst possible time). I hate doing this but I will if necessary (ask around).

This is intended to be a fairly comprehensive list of policies and provisions but something may have been missed; other policies or changes to existing policy may be announced and will supersede any conflicting statements made here.

Chapter 2

Probability

2.1 Definitions

- Experiment: Some action that takes place in the world
- Outcomes = Sample Space = Ω = the universe, every *basic outcome* that could happen
- Event = $A \subseteq \Omega$ = something that happened (could be more than one basic outcome)
- Probability Distribution = $P : \Omega \rightarrow [0, 1]$, $\sum_{x \in \Omega} P(x) = 1$, i.e. values sum to 1 and no value is negative

2.2 Example

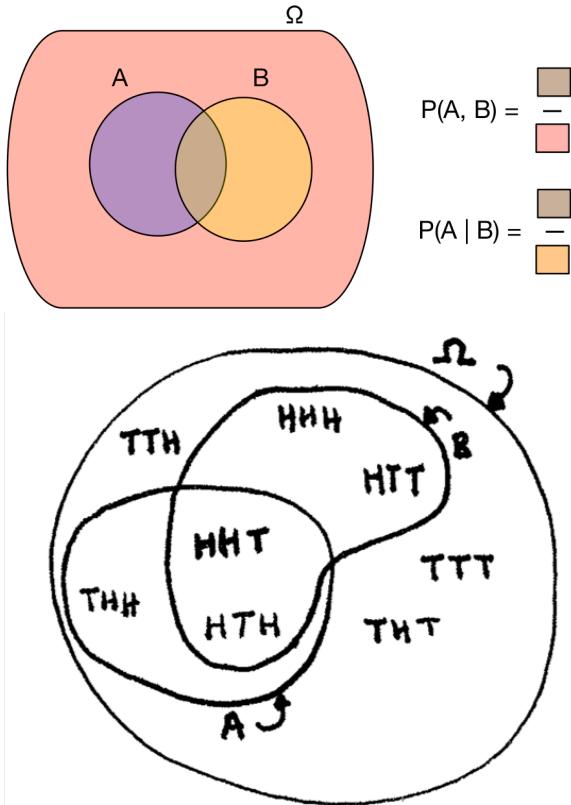
- Experiment = “toss a coin three times”
- $\Omega = \{HHH, HHT, HTT, HTH, THH, THT, TTT, TTH\}$
- Event A = “exactly two heads” = $\{HHT, HTH, THH\}$
- Event B = “first one was heads” = $\{HHH, HHT, HTT, HTH\}$
- Distribution: assign a number between 0 and 1 (‘probability’) to each basic outcome¹; sum of all such numbers = 1
- Uniform Distribution: define $P(x) = c \forall x \in \Omega$...in this case?
- Probability of an event = sum of the probability of its basic outcomes
- So, $P(A) = ?$ and $P(B) = ?$

¹actually to each event in a partition but we’ll get back to that in a minute

2.3 Joint and Conditional Probability

$P(A, B) = P(A \cap B)$ = ‘joint probability of A and B’, i.e. probability of the event formed by the intersection operation (can think of it as probability of ‘the joint event’)

$P(A|B) = \frac{P(A \cap B)}{P(B)}$ = ‘conditional probability of A given B’, i.e. the joint event above, assuming that B is Ω



So $P(A) = 3/8$, and $P(B) = 1/2$

$P(A, B) = P(A) + P(B)$? (No. What is it?)

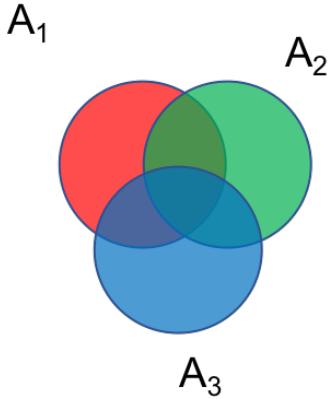
$P(B|A) =$ “if you’ve got two heads what’s the chance your first was heads” = ?

$P(A|B) =$ “if your first is heads what’s the chance you’ve got two” = ?

2.4 Chain Rule of Probability

(Not to be confused with the chain rule of Calculus)

Since $P(A|B) = \frac{P(A,B)}{P(B)}$ (by definition), we can rewrite terms to get $P(A, B) = P(A|B)P(B)$.



Now consider three events, A_1, A_2, A_3 . How can we define $P(A_1, A_2, A_3) = P(A_1 \cap A_2 \cap A_3)$ in terms of conditional probabilities?

Recall that an event is just a set of basic outcomes. So let's define a new event

$$A_{23} = A_2 \cap A_3$$

Then we would write

$$P(A_1, A_{23}) = P(A_1|A_{23})P(A_{23})$$

Now, substitute back in the joint event that A_{23} represents

$$P(A_1, A_2, A_3) = P(A_1|A_2, A_3)P(A_2, A_3)$$

Now substitute the definition of joint probabilities in terms of conditional probabilities again

$$P(A_1, A_2, A_3) = P(A_1|A_2, A_3)P(A_2|A_3)P(A_3)$$

Of course $P(A_1, A_2, A_3) = P(A_3, A_2, A_1)$ (set intersection is commutative) so you could write this instead as $P(A_3|A_2, A_1)P(A_2|A_1)P(A_1)$

The general chain rule for probabilities is:

$$P(A_1, \dots, A_N) = P(A_1|A_2, \dots, A_N) \times \dots \times P(A_{N-1}|A_N) \times P(A_N)$$

2.5 Independence

A and B are *independent* if the occurrence of one does not affect the occurrence of the other, i.e. if $P(A|B) = P(A)$. Corollary: $P(B|A) = P(B)$. Corollary: $P(A, B) = P(A)P(B)$.

Exercise: Prove that these three statements are corollaries of each other.



2.6 Bayes' Rule/Theorem/Law

$P(A|B) = \frac{P(A,B)}{P(B)}$, by definition. Thus, $P(A, B) = P(A|B)P(B)$.

Because intersection is commutative (see above), $P(A, B) = P(B|A)P(A)$. This also explains the corollary noted in Section 2.5. This leads to Bayes' Rule/Theorem/Law:

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

This can be very helpful when you have information about one conditional direction but you want info about the other direction.

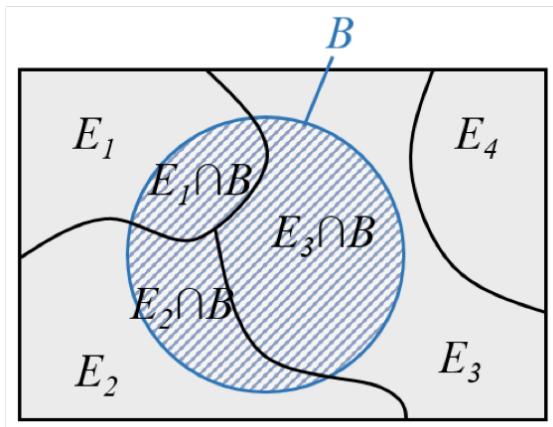
2.7 Law of Total Probability

We say events E_1, \dots, E_n partition Ω if:

$$\forall i, j \in [1, n], E_i \cap E_j = \emptyset$$

and

$$\sum_{i=1}^n P(E_i) = 1$$



The Law of Total Probability says, given partitioning events $E_1 \dots E_n$ and event B :

$$P(B) = \sum_{i=1}^n P(B, E_i)$$

2.8 Example

Some people can read minds, but not many: $P(MR) = 1/100,000 = .00001$.

There is a test to read minds; if you are a mind reader I can detect this very well: $P(T|MR) = 0.95$ and if you're not I can detect this even better: $P(\neg T|\neg MR) = 0.995$. Note: $\{T, \neg T\}$ partition the event space, as do $\{MR, \neg MR\}$.

If Jill gets a positive result on the test, how likely is it she is the mind reader?

i.e., $P(MR|T) = ?$

By Bayes' Law, $P(MR|T) = \frac{P(T|MR)P(MR)}{P(T)}$.

We need to get $P(T)$. By law of total probability, $P(T) = P(T, MR) + P(T, \neg MR)$. By definition of conditional probability, $P(T, MR) = P(T|MR)P(MR) = 0.95 \times .00001$; $P(T, \neg MR) = P(T|\neg MR)P(\neg MR) = 0.005 \times .99999$. $P(T) = .00500945$ and $P(MR|T) \approx .002$

Chapter 3

Ethics Intro

3.1 Why Discuss Ethics in NLP Now?

A lot (all?) of this material comes from UW and CMU courses in NLP ethics. This is not a full course in the subject, so much will be skipped.

See http://demo.clab.cs.cmu.edu/ethical_nlp/ and http://faculty.washington.edu/ebender/2017_575/index.html.

We just discussed probability...this seems like a topic jump. It's important as we get into the actual details to consider the ethical implications of:

- What we create (power of the models and tasks to do good or harm)
- How we create it (data sets and other ways that biases are implicitly baked into models)
- Why we create it (who is funding the work? How will it be used after it's made?)

We are dealing with *human language* which means we are dealing with people.

Nice quote: "The common misconception is that language has to do with words and what they mean. It doesn't. It has to do with people and what they mean." (Herbert H. Clark and Michael F. Schober, 1992)

Ethics is broadly 'what is good/right.' However, what is good or right? Squishy!!

3.1.1 Example from CMU overview

- Should we design a classifier to predict if a chicken is male or female, while in the egg?
 - + Lowers cost of hatching/raising chicks you don't want (females for eggs, males for meat)
 - + Destroying the eggs may be less painful than killing the unwanted chicks
 - - it's not the chicken's fault
 - ...
- Should we design a classifier to predict IQ of an adult? Of a child? Of a fetus?

- Positives? Negatives?
- Let's stick with adults for now.
 - Who benefits from this classifier?
 - Who can be harmed, even if the classifier is *never wrong*?
 - If the classifier is more accurate for e.g. white women than other groups, who is responsible for the failings? Developer? Manager/Reviewer? University/Company? Society?
- NLP increasingly used to make real-world decisions.
 - Credit-worthy
 - Recommendations (based on some kind of stereotypes)
 - self-driving car decisions (more vision than NLP but commands)
 - whether to grant parole
- Data collection and annotation is a big part of NLP. Human language necessarily means human collection
 - Unsupervised data: intentional publication (news)? Unintentional publication (text messages)? Semi-intentional publication (social media)? Monetized publication (copyright violation)?
 - Annotation: Is this considered human subjects research (HSR)? Can it be distressing to annotators?
 - Are providers of data being fairly compensated? (Mechanical Turk, scraping against copyright, etc.)
 - Are these limits altering the demographic distribution of the data, and what are the consequences with regard to model performance?
- Issues to consider throughout this course:
 - How can this be used?
 - How might this be used?
 - What are the consequences of this use? Who will be affected? Who won't be able to take advantage?
 - Who has interest (ownership or otherwise) of the data you will use?
- The answer is probably not 'abandon everything you are doing here.' And I don't have all the answers (and in many cases there is no answer). But it is important to be aware of these issues.

Chapter 4

Corpora Processing and Linear Classifiers (Naive Bayes)

4.1 History of Methodologies

4.1.1 Pre-Statistical (1650s/1950s through approx. 1980s)

- Mostly about modeling specific linguistic phenomena in a small number of sentences, sometimes using code
- Linguists/highly trained coders wrote down fine-grained detailed rules to capture various aspects, e.g. ‘“swallow” is a verb of ingestion, taking an animate subject and a physical object that is edible...’
- Very time-consuming, expensive, limited coverage (brittle), but high precision
- Academically satisfying, but not good at producing systems beyond the demo phase

4.1.2 Statistical

- Empirical approach: learn by observing language as it's used “in the wild”
- Many different names:
 - Corpus Linguistics
 - Empirical NLP
 - Statistical NLP
- Central tool:
 - corpus
 - thing to count with (i.e. statistics)
 - (later on) machine learning methodologies, software/hardware for helping with scale

- Advantages
 - Generalize patterns as they actually exist (i.e. bottom-up, not top-down)
 - Little need for knowledge (just count)
 - Systems are robust and adaptable (change domain by changing corpus)
 - Systems degrade more gracefully (corner cases captured in data)
 - Evaluations are (more) meaningful

- Limitations

- Bound by data – can't model what you can't see – “I held the book with my arm stretched out and opened my hand. It (floated away), (fell to the ground)”
- Big Data methods fail when the data is small or wrong – sometimes you want to try to translate Oromo news to English with 50,000 words of bible when you want 10m+words of news
- more computationally expensive (but less human-expensive) (usually a good trade-off)
- Methods don't have the same pattern-recognition and generalization abilities of humans learning (and putting into rule-based methods) which can lead to unintuitive brittleness.

4.1.3 Corpus (pl: corpora): a collection of (natural language) text systematically gathered and organized in some manner

- Features:
 - Size
 - Balanced/domain
 - Written/Spoken
 - Raw/Annotated
 - Free/Pay
- Some Famous (text) examples:
 - Brown Corpus: 1m words balanced English text, POS tags
 - Wall Street Journal: 1m words English news text, syntax trees
 - Canadian Hansards: 10m words French/English parliamentary text, aligned at sentence level
 - Wikipedia (all of it): 3b words
 - Google books ngrams: 500B words
 - Common Crawl: 1T words
 - Any others of particular interest?

4.1.4 How Big Does it need to be?

- We'd like to get examples of all linguistic phenomena, ideally several times so we know how likely they are to occur
- How big should a corpus be to get every possible sentence in English?
 - Every possible idea?
 - Every 5-word phrase?
 - Every word?
- None of these are possible!

4.1.5 corpus processing

```
# word counts and ngram counts

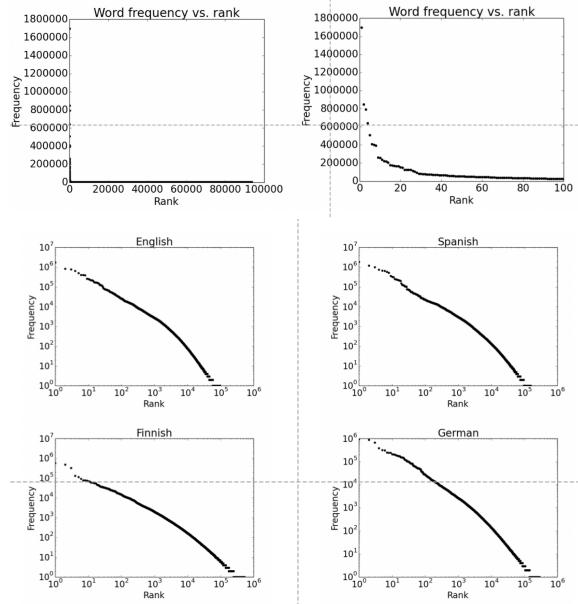
# 10 most frequent words in the text
sed 's/ /\n/g' sawyr11.txt | sort | uniq -c | sort -k1nr | head
# 10 most frequent words in the text after removing blank lines
sed 's/ /\n/g' sawyr11.txt | grep -v "^$" | sort | uniq -c | sort -k1nr | head
# 10 most frequent bigrams (2 word sequences) in the text

sed 's/ /\n/g' sawyr11.txt | grep -v "^$" > ts.words
tail -n+2 ts.words > ts.2pos
paste ts.words ts.2pos | sort | uniq -c | sort -k1nr | head
# count number of words/ngrams, number of word/ngram types, number of
# 1-count word/ngram types

# words in the text without blank lines, one word per line, saved to a file
# (for convenience)
sed 's/ /\n/g' sawyr11.txt | grep -v "^$" > ts.words
# number of word tokens
wc -l ts.words
# number of word types
sort ts.words | uniq | wc -l
# number of one-count words
sort ts.words | uniq -c | awk '$1==1{print}' | wc -l
# number of two-word sequence (bigram) tokens
# (based on the answer to the number of word tokens you should know this
# without running the command...
paste ts.words <(tail -n+2 ts.words) | wc -l
# number of bigram types
paste ts.words <(tail -n+2 ts.words) | sort | uniq | wc -l
# number of one-count bigrams
paste ts.words <(tail -n+2 ts.words) | sort | uniq -c | awk '$1==1{print}' | wc -l
```

4.1.6 Zipf's law

Take a naturally occurring corpus (in this case, of text). Count frequency of words. Order words by frequency, to form ranks. Then the rank of a word is inversely proportional to its frequency.



For frequency f and rank k , $f \approx \frac{1}{k^a}$ for some constant a . Thus $-a \approx \frac{\log(f)}{\log(k)}$. Consequences:

- There will always be a lot of infrequent or unseen words
- This is true at all levels of linguistic structure
- So we have to find clever ways of generalizing so we can get reasonable estimates for things we haven't seen often enough

4.2 Nominal task (also HW1): text classifier

We have some book reviews. We want to know automatically if they're positive or negative.
Positive review:

I loved this book. The food was really good and fast (which is good because I have a very packed schedule). Also great if you're on a budget. The recipes have variations so I could eat different things but not have to buy a whole new set of ingredients.

Negative review:

I tried reading this book but found it so turgid and poorly written that I put it down in frustration. It reads like a translation from another language by an academic bureaucrat. The theme is interesting, the execution poor. Cannot recommend

4.2.1 Evaluation

We should always ask ‘how do we know we are doing well at what we are trying to do?’ This time the answer is fairly simple (it won’t always be). We collect many reviews along with their labels (Positive/Negative, which we may have to create out of some other labels, like a numerical score).

We’ll calculate simple *accuracy*: $\frac{\# \text{ correct}}{\# \text{ total}}$.

4.2.2 Data

We will divide our data into *train*, *development* (dev) (also sometimes called ‘validation’) and *test* corpora. *train* is used to build a model and is the largest data set (usually 80% of the data). *dev* is not used to train but is frequently consulted during optimization (where relevant) to avoid *overfitting* on training data. *test* is usually not evaluated or looked at except for when optimization is done, to ensure even less overfitting. Sometimes an even more super-secret *blind* test set is not even available to you but held off to test your final model.

4.2.3 Framework

Here’s a simple framework for this problem:

```
import operator
def evaluate(sentence, option, model):
    # fill me in
    pass

def classify(sentence, options, model):
    scores = {}
    for option in options:
        scores[option] = evaluate(sentence, option, model)
    return max(scores.items(), key=operator.itemgetter(1))[0]
```

Let’s consider methods for the ‘evaluate’ function:

4.3 Rule-Based (top-down?) Model

Positive reviews should have positive words, negative reviews should have negative words. Thankfully, people have compiled such *sentiment lexicons* for English. See <http://www.enchantedlearning.com/wordlist>.

Positive words example: {absolutely, adorable, bountiful, bounty, cheery}

Negative words example: {angry, abysmal, bemoan, callous}

Let’s put the intuition and data together:

```
# externally constructed; don't actually structure your code this way
```

```

# probably shouldn't structure your code this way...
model = {}
model[ 'good' ] = set([ 'yay', 'love', ...])
model[ 'bad' ] = set([ 'terrible', 'boo', ...])

def evaluate(sentence, option, model):
    score = 0
    for word in sentence.split():
        if word in model[option]:
            score+=1
    return score

```

4.4 Empirical Model

Our intuitions about word sentiment aren't perfect and neither are those of the people who made the word list. But we do have many examples of reviews and their sentiments. So we can make our own list of good and bad words. Instead of hard-coding the model as I did above, we can create a *training* function that takes in sentences *with their labels* and then returns a model:

```

from collections import Counter, defaultdict
def train(labeled_sentences):
    scores = defaultdict(lambda: Counter()) # doubly nested structure
    for sentence, label in labeled_sentences:
        for word in sentence.split():
            scores[word][label]+=1
    model = defaultdict(lambda: set())
    for word, table in scores.items():
        # the most frequent label associated with the word
        label = max(table.items(), key=operator.itemgetter(1))[0]
        model[label].add(word)
    return model

```

We now have our first *supervised* model. We should back up and consider what we are actually trying to model from a probabilistic view. For one thing let's consider the actual probability of each label, rather than our ad-hoc adding method above.

Our classifier should choose $\text{argmax}_y P(y|s)$ for sentence s where y is one of a fixed set of labels. But we didn't consider s as some monotone thing; we considered the occurrence of each word as an event.

So we'll make an assumption called the *bag of words assumption* which is that for sentence $s = w_1 w_2 \dots w_n$, we say $P(y|s) = P(y|w_1, w_2, \dots, w_n)$. Note this now doesn't depend on the order of the words.¹

¹This assumption isn't really part of Naive Bayes, it's an assumption about the feature set being used. It's probably better to say that for x, y , we calculate $f(x, y) = f_1, \dots, f_n$ and then proceed from there. The above is a bit of a simplification. The reading avoids this simplification.

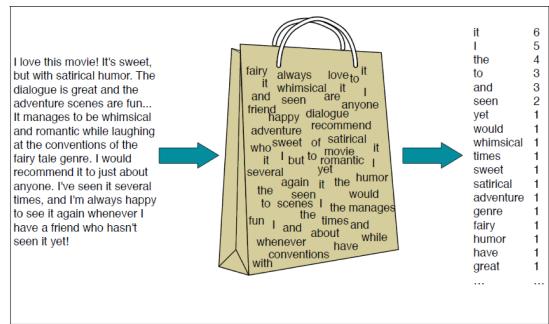


Figure 7.1 Intuition of the multinomial naive Bayes classifier applied to a movie review. The position of the words is ignored (the *bag of words* assumption) and we make use of the frequency of each word.

Figure from J&M 3rd ed. draft, sec 7.1

But this model will only be valid if we can calculate probabilities of a label for the exact multiset of each sentence's words. We need another assumption, the *Naive Bayes assumption* which is that the probability of a label given a word is conditionally independent of the other words.

Note from Bayes' rule:

$$P(y|w_1, w_2, \dots, w_n) = \frac{P(w_1, w_2, \dots, w_n|y)P(y)}{P(w_1, w_2, \dots, w_n)}$$

and since the word sequence itself is constant, we can say

$$\text{argmax}_y P(y|w_1, w_2, \dots, w_n) = \text{argmax}_y P(w_1, w_2, \dots, w_n|y)P(y)$$

The conditional probability assumption, which makes up the Naive Bayes assumption, can then be applied, so we assume

$$P(w_1, w_2, \dots, w_n|y)P(y) = P(w_1|y)P(w_2|y), \dots, P(w_n|y)P(y)$$

Is this a good model? George Box, statistician: “All models are wrong, but some models are useful.”

What are problems with the bag of words assumption?

What are problems with the naive bayes assumption?

Does it work? Yes, for many tasks actually. And it's very simple so it's usually worth trying.

Here's a new trainer:

```
from collections import Counter, defaultdict
def train(labeled_sentences):
    wscores = defaultdict(lambda: Counter()) # doubly nested structure
    cscores = Counter()
    for sentence, label in labeled_sentences:
        for word in sentence.split():
            wscores[label][word] += 1
            cscores[label] += 1
    model = { 'cprobs':{}, 'wprobs':{} }
    for label in cscores.keys():
        model['cprobs'][label] = cscores[label]
        model['wprobs'][label] = { word: wscores[label].get(word, 0) / cscores[label] for word in wscores[label].keys() }
```

```

model[ 'cprobs' ].append( cscores[ c ] / len( labeled_sentences ) )
wprob = {}
for word, score in wscores[ label ]:
    wprob[ word ] = score / cscores[ label ]
model[ 'wprobs' ].append( wprob )
return model

```

And a new, more appropriate classifier

```

def evaluate(sentence, option, model):
    score = model[ 'cprobs' ][ option ]
    for word in sentence.split():
        score *= model[ 'wprobs' ][ option ][ word ]
    return score

```

4.4.1 Practicalities: smoothing

$score *= model['wprobs'][option][word]$ is going to be problematic if we have never seen a word with a particular class. Solution: smoothing!

Laplace (add-1) smoothing: assume you've seen every word (even words you haven't seen before) with every class!

Before (assume 10k words in training set of negative items):

$P(\text{amazing}|\text{negative}) = 0/10,000 = 0$ (seen the word but not with class 'negative')

$P(\text{blargh}|\text{negative}) = 0/10,000 = 0$ (never seen the word)

Introduce a new term, 'OOV', and if you haven't seen your test word during training, pretend your word is 'OOV'. Then, since you add 1 for each vocabulary word and the OOV with each class, if your vocabulary size was 500, you now get:

$P(\text{amazing}|\text{negative}) = 1/10,501$

and

$P(\text{blargh}|\text{negative}) = 1/10,501$

You may want to actually *introduce* some OOV into your training set, by replacing words that appear fewer than some k times with OOV. This is so that your model can learn how to behave with OOVs.

4.4.2 Practicalities: underflow

Recall:

```

for word in sentence.split():
    score *= model[ 'wprobs' ][ 'option' ][ 'word' ]

```

Sentence may be long! Probabilities may be small! It's very easy to run into underflow: try this:

```

a=1
for i in range(100):
    a*=.0001
    if i % 10 == 0:
        print(i, a)

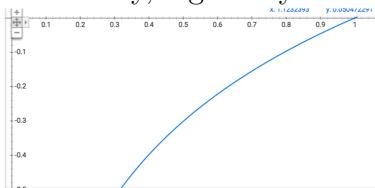
```

```

0 0.0001
10 1.000000000000003e-44
20 1.000000000000007e-84
30 1.000000000000001e-124
40 1.0000000000000015e-164
50 1.0000000000000021e-204
60 1.0000000000000029e-244
70 1.0000000000000036e-284
80 0.0
90 0.0

```

Thankfully, logs are your friend, because of the following graph of $\log(x)$:



So instead rewrite the function as

```

from math import log

def evaluate(sentence, option, model):
    score = log(model['cprobs'][option])
    for word in sentence.split():
        score += log(model['wprobs'][option]['word'])
    return score

```

(Note the operator change)

Chapter 5

Linear Classifiers Continued (Perceptron & Logistic Regression)

5.1 problems with naive bayes assumption

We tolerated the naive bayes assumption: $P(w_1, w_2, \dots, w_n|y)P(y) = P(w_1|y)P(w_2|y), \dots, P(w_n|y)P(y)$ because it was useful, but we know that it is incorrect. Let's take a closer look:

So, by chain rule, $P(w_1, w_2, \dots, w_n|y) = P(w_1|y, w_2, \dots, w_n)P(w_2|y, w_3, \dots, w_n) \dots P(w_n|y)$ but by the Naive Bayes assumption we restrict the conditional to just be y .

Of course, this is wrong! Some words are clearly not conditionally independent of each other, i.e. $P(\text{San}|y) \neq P(\text{San}|y, \text{Francisco})$.

A more intuitive example: Imagine if 9/10 people recommended a movie to you. What if 8 of those 9 didn't actually see the movie but just repeated whatever the 9th person said to you?

Naive Bayes may in fact be a decent assumption for the specific case we've seen it in, and for most other words that occur alongside it (except for nearby words), but this is not the only *feature* we might care about. In particular we may care about overlapping features (e.g. the word, the word AND its predecessors/successors, prefix of the word, if the word is on certain lists of words, etc.

5.2 Representation of features and weights for multi-class classification (esp. in Eisenstein)

We're going to talk about features in general but more importantly we're going to talk about feature weights, and especially if you've taken ML before you may be a bit confused by the notation in Eisenstein that I will borrow:

Sentence x : 'This movie rocks'

Label possibilities: 'Positive (+), Negative (-), Neutral (\emptyset)'

Feature classes: "number of words" (nW), "contains 'happy'" (ch), "contains word ending in s" ($*s$)

Feature function f :

$$\begin{aligned}f(x, +) &= [3, 0, 1, 0, 0, 0, 0, 0, 0] \\f(x, \emptyset) &= [0, 0, 0, 3, 0, 1, 0, 0, 0] \\f(x, -) &= [0, 0, 0, 0, 0, 0, 3, 0, 1]\end{aligned}$$

For every feature we assign a weight. In the previous Naive Bayes discussion the features were all of the form ‘contains x’ for word x (could actually be ‘number of times we see x’), and the weights were $P(x|y)$ (from the Naive Bayes assumption). As we see above, the features can be more arbitrary than that. We can arrange our weights in a weight vector, which by convention we’ll call θ :

$$\theta = [P(nW|+), P(ch|+), P(*s|+), P(nW|\emptyset), P(ch|\emptyset), P(*s|\emptyset), P(nW|-), P(ch|-), P(*s|-)]$$

So given f and θ we can do ‘inference’ like so:

$$\text{argmax}_{y \in \mathcal{Y}} \theta \cdot f(x, y)$$

Eisenstein uses $\Psi(x, y) = \theta \cdot f(x, y)$; I call that the ‘model score’ or ‘model cost’; it’s the model’s opinion of y being suitable for x .

What about $P(y)$? Note this is the background probability of the class y . We can include this as a term that is always on for the set of features associated with class y . It’s called the ‘bias’. So revising the above,

$$f(x, +) = [3, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0] \text{ and so on...}$$

$$\theta = [P(nW|+), P(ch|+), P(*s|+), P(+),$$

$$P(nW|\emptyset), P(ch|\emptyset), P(*s|\emptyset), P(\emptyset),$$

$$P(nW|-), P(ch|-), P(*s|-), P(-)]$$

5.3 perceptron

I called θ ‘weights’ for good reason – who says they have to be probabilities? We can adopt a ‘trial and error’ approach:

```
theta = random weights
```

```
for each sentence, label in data:
    if we would choose some other label over the correct one:
        modify theta so we don't do that
return theta
```

Here it is in a bit more gory detail using the framework from before:

```
import numpy as np
```

```
# note that evaluate needs to be rewritten to be more general;
# left as an exercise to the reader, but should include a features() method

def train(labeled_sentences, options, featsize):
    # should be a better way to determine feat size
    # probably want to initialize differently
    model = {'theta': np.random(featsize)}
    for i in range(iterations): # user-determined
```

```

for sentence , label in labeled_sentences :
    hyp = classify(sentence , options , model)
    if hyp != label :
        model[ 'theta' ] += features(sentence , label)–features(sentence , hyp)
return model

```

Quick illustration: Some feature type f_1 has value 2, f_2 has value 1, f_3 has value 0.

feats	$f(x, 1)$	$f(x, 2)$	θ
f_{11}	2	0	0.3
f_{21}	1	0	0.7
f_{31}	0	0	0.8
f_{12}	0	2	-0.2
f_{22}	0	1	2.2
f_{32}	0	0	-0.4

Let's say class 1 is correct. Given the table above, we get the following model costs:

$$\Psi(x, 1) = 1.3; \Psi(x, 2) = 1.8.$$

So update $\theta = \theta + f(x, 1) - f(x, 2)$:

feats	$f(x, 1)$	$f(x, 2)$	θ
f_{11}	2	0	2.3
f_{21}	1	0	1.7
f_{31}	0	0	0.8
f_{12}	0	2	-2.2
f_{22}	0	1	1.2
f_{32}	0	0	-0.4

Class 2's weights went down (if they affected the outcome) and class 1's went up. Now we get the following model costs:

$$\Psi(x, 1) = 6.3; \Psi(x, 2) = -3.2.$$

So the item is correctly classified.

Some things to discuss in the context of machine learning: averaging all θ at the end, learning rates, batch sizes.

5.4 loss function justification for perceptron

This seemed to work in the demo case above but has a very ad-hoc feel to it. (Side note: often times models *are* originally designed in an ad-hoc way and only later is the theory worked out. The original paper on perceptrons from 1957 has, AFAICT, nothing regarding the following) Why does this work?

It's worth considering the *loss* of the model. Remember, the model conveys an opinion about how to classify¹ that is to some degree untrue. Loss can be thought of as 'how wrong the model is.' For perceptron the loss for item i in a data set is (from Eisenstein):

$$L_{\text{perceptron}}(\theta; \mathbf{x}^{(i)}, y^{(i)}) = \max_{y \in \mathcal{Y}} \theta \cdot f(\mathbf{x}^{(i)}, y) - \theta \cdot f(\mathbf{x}^{(i)}, y^{(i)})$$

¹or for regression models, the value associated with an input

The first term ($\max_{y \in \mathcal{Y}} \theta \cdot f(\mathbf{x}^{(i)}, y)$) is exactly how we pick a label, and the second term is the model score of the right label. If we picked the right label, the loss is zero. Otherwise, the model score for the wrong label is higher, and the amount higher is how wrong we are.

Why do we have loss²? Because there's something wrong with our model, i.e. θ . But we can change that. How much should we change it? To minimize the loss, of course!

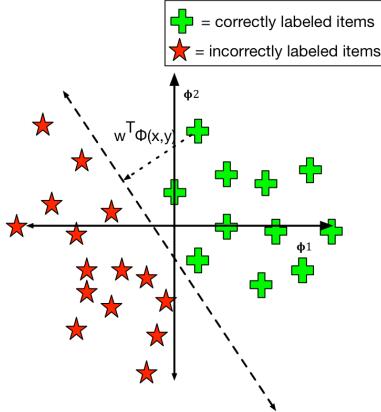
We can take the derivative of L with respect to θ . By adjusting θ in the negative direction of the gradient we will have a lower loss on our training data.

Let's assume \hat{y} is the hypothesis y and that it's not the true label; what's the derivative? Well what's the equation?

Let's call $\hat{\mathbf{f}} = f(\mathbf{x}, \hat{y})$ and $\mathbf{f} = f(\mathbf{x}, y)$ (I dropped the superscripts; too hard to type) and make them subscriptable. Then $L = \theta_1 \hat{f}_1 - \theta_1 f_1 + \theta_2 \hat{f}_2 - \theta_2 f_2 + \dots + \theta_n \hat{f}_n - \theta_n f_n$. Then $\partial L / \partial \theta_1 = \hat{f}_1 - f_1$ or to be more vector wise about it, $\partial L / \partial \theta = \hat{\mathbf{f}} - \mathbf{f}$. So the negative of that is $\mathbf{f} - \hat{\mathbf{f}}$.

You might wonder 'isn't there a closed form of this?' Yes, there is, basically you need to take the inverse or pseudo-inverse of your feature matrix (feature vector for each data item). Why isn't it used? Because the matrix inversion becomes really slow ($O(n^2)$ in general) once the data gets large.

BTW, another way to look at what the perceptron is doing is finding the *separating hyperplane* between correctly and incorrectly labeled samples. This is easiest to visualize in the binary case:



The weights define a line/plane/hyperplane along which the score is zero; we want all the correctly and incorrectly labeled examples to be divided by that separator.

5.5 logistic regression

One nice thing about the naive bayes model is that it's probabilistic, so if your classifier is one part of a pipeline, you can tell the rest of the pipeline your confidence in your output in a rational way. $\Psi(x, y)$ has range $(-\infty, \infty)$ which is less helpful. Another issue with perceptron is it only is concerned with making sure the correct answer is chosen (in training).

Nobody forced us to keep $\Psi(x, y)$ as the model score. A preferred model score would be $P(y|x)$, the conditional probability of the output given the input. How do we form a probability distribution from a set of scores?

²Assuming our data is *separable*, which see below

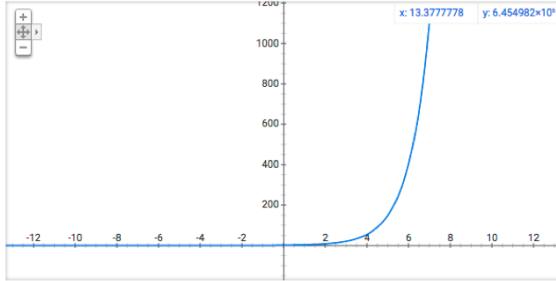
We can't simply normalize:

$$\frac{\Psi(x, y)}{\sum_{y' \in \mathcal{Y}} \Psi(x, y')}$$

will not work. Why?

Instead we can use e^{Ψ} . why?

Graph for $\exp(x)$



So our new model cost will be

$$\frac{e^{\Psi(x, y)}}{\sum_{y' \in \mathcal{Y}} e^{\Psi(x, y')}} \quad (5.1)$$

This, by the way, is the ‘softmax’ function that comes up a lot in neural networks. It does exactly the same job we are trying to do here: convert a set of numbers in the range $(-\infty, \infty)$ into a distribution while keeping the ordering the same. In fact, it generally ‘peaks’ the highest number; that’s why this can be thought of as a ‘soft’ form of the ‘max’ operator (hence the name).

Now that we’ve got a distribution we have a natural loss function we can use: cross-entropy! $H(p, q) \triangleq -E_p \log q$, where E_p is the *expected value* w/r/t p . Concretely:

$$H(p, q) = - \sum_{x \in \mathcal{X}} p(x) \log q(x)$$

Where here, $q(x) = q(y|x)$ = Equation 5.1. From information theory this is the average number of bits³ needed to identify an item if q is used to identify the item, but p is the true distribution. So intuitively you want as small a cross-entropy as possible, and that makes this a natural loss function. But what is $p(y|x)$, the true distribution? A good choice for supervised learning is to assume that the provided label is the truth and should always occur and that all other labels should never occur. That means that our calculation of cross entropy, which will in practice use training set \mathcal{X} and label space \mathcal{Y} :

$$-\frac{\sum_{x,y \in \mathcal{X}} \sum_{y' \in \mathcal{Y}} p(y'|x) \log q(y'|x)}{|\mathcal{X}|}$$

becomes

$$-\frac{\sum_{x,y \in \mathcal{X}} \log q(y|x)}{|\mathcal{X}|}$$

³really, nats

since $p(y|x) = 1$ and $\forall y' \in \mathcal{Y} \setminus y, p(y'|x) = 0$.

Now considering a single item and substituting back in Equation 5.1:

$$\begin{aligned} & -\log \frac{\exp(\theta \cdot f(\mathbf{x}, y))}{\sum_{y' \in \mathcal{Y}} \exp(\theta \cdot f(\mathbf{x}, y'))} \\ & -\theta \cdot f(\mathbf{x}, y) + \log \sum_{y' \in \mathcal{Y}} \exp(\theta \cdot f(\mathbf{x}, y')) \end{aligned}$$

Now, the gradient:

$$\begin{aligned} \partial L / \partial \theta &= -f(\mathbf{x}, y) + \frac{1}{\sum_{y'' \in \mathcal{Y}} \exp(\theta \cdot f(\mathbf{x}, y''))} \sum_{y' \in \mathcal{Y}} \exp(\theta \cdot f(\mathbf{x}, y')) f(\mathbf{x}, y') \\ &= -f(\mathbf{x}, y) + \sum_{y' \in \mathcal{Y}} \frac{\exp(\theta \cdot f(\mathbf{x}, y'))}{\sum_{y'' \in \mathcal{Y}} \exp(\theta \cdot f(\mathbf{x}, y''))} f(\mathbf{x}, y') \\ &= -f(\mathbf{x}, y) + \sum_{y' \in \mathcal{Y}} q(y'|x; \theta) f(\mathbf{x}, y') \\ &= -f(\mathbf{x}, y) + E_q f(\mathbf{x}, y') \end{aligned}$$

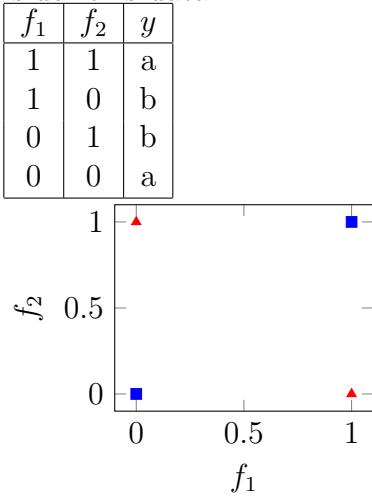
We update by the negative of the gradient, i.e. $f(\mathbf{x}, y) - E_q f(\mathbf{x}, y')$. Intuitively, we are now considering not only how far away each wrong answer is from the right answer, but how confident we are about each wrong answer. If we have low confidence about an answer it will affect the loss very little.

Chapter 6

Nonlinear Classifiers

6.1 Why Nonlinear Models?

The linear models we introduced appear to be very flexible, however they are limited in what they can capture. Specifically, because the equation $\theta \cdot f(x, y)$ is *linear*, classification cannot be successful if the data points, when plotted in their feature space, cannot be divided by a line (or, more generally, a hyperplane). The classic example of this is the *xor problem*. Consider this data:



This 2-label data set is class 1 iff binary features f_1 and f_2 are both on or both off and is class -1 otherwise. Try to draw a line that separates the data. It of course can't be done. You could of course introduce a new feature $\text{XOR}(f_1, f_2)$ that explicitly captures this relationship and then the data would be linearly separable. But in general you don't know which combinations of features yield separability, which .

You could try a transformation that makes combinations of the weights. Define weights w_{11}, w_{21}, b_1 to map from the old feature space to a new feature g_1 and w_{12}, w_{22}, b_2 to map from the old feature space to a new feature g_2 , such that

$$\begin{aligned}g_1 &= w_{11}f_1 + w_{21}f_2 + b_1 \\g_2 &= w_{12}f_1 + w_{22}f_2 + b_2\end{aligned}$$

Let's use these as the weights:

$$\begin{bmatrix} 1(w_{11}) & -1(w_{12}) \\ 1(w_{21}) & -1(w_{22}) \end{bmatrix}$$

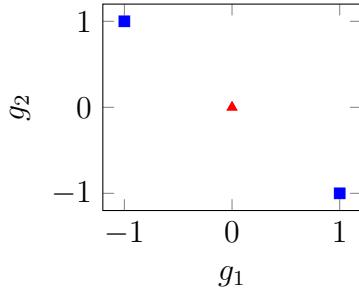
and

$$\begin{bmatrix} -1(b_1) & 1(b_2) \end{bmatrix}$$

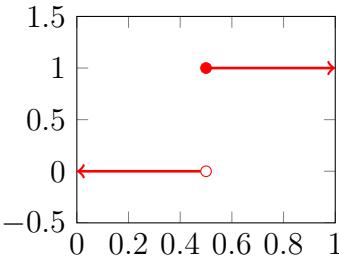
(It's no accident I set these up as a matrix)

That yields:

g_1	g_2	y
1	-1	a
0	0	b
0	0	b
-1	1	a

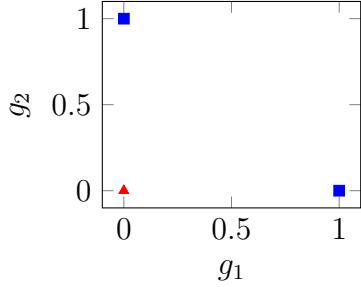


It's still non-separable! This should be no surprise; all a linear transformation can do is scale, transpose, and rotate the points; it can't distort them in a way that allows separability. So we'll multiply by a non-linear step function:



g_1	g_2	y
1	0	a
0	0	b
0	0	b
0	1	a

Separable!



The point of nonlinear transformations is to enable *recombinations* of features. We can make a linear combination of the new features and apply a nonlinearity to get yet another recombination. This can be done as many times as needed. What's nice about this is that *we don't need to specify complicated features* any more – if we choose weights properly and use enough layers we can capture any combinations of the input data.

6.1.1 Obtaining the weights

In logistic regression and perceptron we used *gradient descent* of the loss on training data to set weights. We can use the same approach here, though the step function, being non-differentiable, isn't an appropriate nonlinear *activation function*, so we'll use a similarly shaped function that is differentiable at every point. First let's define the model and the loss. Let \mathbf{x}, y be the input feature vector and its label. Let \mathbf{H} be the weights matrix and \mathbf{b}_H be the bias vector¹. The elementwise nonlinear activation function is $g()$. Thus to get the transformed vector (or ‘hidden’ features...or even ‘hidden vector’) \mathbf{h} :

$$\begin{aligned}\mathbf{z} &= \mathbf{x}\mathbf{H} + \mathbf{b}_H \\ \mathbf{h} &= g(\mathbf{z})\end{aligned}$$

What are the lengths of \mathbf{x} and \mathbf{h} ? That's up to you to some degree. Usually the inputs for \mathbf{x} have something to do with the words of the original input.² We could stick with naive Bayes features and say \mathbf{x} is $|V|$ (vocabulary size)-dimensional, let's say 50,000, with a count of frequency in the text of each word. $|\mathbf{h}|$ is entirely a design decision. Let's pick 1000. That informs the dimensions of \mathbf{H}, \mathbf{b}_H , and everything else calculated above.

We now need to convert into the output space, which should be equal in length to the number of choices (e.g. for sentiment it could be 2, 3, 5, depending on how the problem is defined. Let's say 5.). And we use softmax again, to ensure the output is probabilistic.

$$\begin{aligned}\ell &= \mathbf{h}\mathbf{O} + \mathbf{b}_O \\ \mathbf{o} &= \text{softmax}(\ell)\end{aligned}$$

The loss L is again the cross-entropy loss, which is defined for one data item (\mathbf{x}, y) as

¹When to use a bias vector? I don't know, and I see different formulations do different things. For example, Eisenstein ch. 3.1 doesn't use bias here but does use bias in the output transform. I will use it in both places

²They are often, in fact, *word embeddings*, but we'll get to that shortly.

$$H_{\mathbf{x},y}(p, q) = - \sum_{y' \in \mathcal{Y}} p(y'|x) \log q(y'|x)$$

where the distribution $q(y'|x)$ may be represented by \mathbf{o} and the true distribution $p(y'|x)$ is one-hot at y , reducing to $-\log(o_y)$.

Thus, L ends up being

$$L = -\log(o_y)$$

i.e. the negative log of the probability of the correct answer (denoted o_y to note that member of \mathbf{o} corresponding to choice y).

Having calculated L , we update each set of parameters $(\mathbf{H}, \mathbf{b}_\mathbf{H}, \mathbf{O}, \mathbf{b}_\mathbf{O})$ by the opposite of the gradient of L with respect to that variable, i.e:

$$\begin{aligned} \mathbf{H} &\leftarrow \mathbf{H} - \lambda \partial L / \partial \mathbf{H} \\ \mathbf{b}_\mathbf{H} &\leftarrow \mathbf{b}_\mathbf{H} - \lambda \partial L / \partial \mathbf{b}_\mathbf{H} \\ \mathbf{O} &\leftarrow \mathbf{O} - \lambda \partial L / \partial \mathbf{O} \\ \mathbf{b}_\mathbf{O} &\leftarrow \mathbf{b}_\mathbf{O} - \lambda \partial L / \partial \mathbf{b}_\mathbf{O} \end{aligned}$$

where λ is a learning rate. Now how are these partials determined? We start at the loss equation itself and use simple calculus:

$$\begin{aligned} L &= -\log(o_y) \\ \partial L / \partial o_y &= -1/o_y \end{aligned}$$

Now consider the definition of o_y itself; we can use the chain rule and the local derivative of o_y with respect to ℓ , though softmax is a slightly tricky function to take a derivative of:

$$\begin{aligned} \partial L / \partial \ell &= \partial L / \partial o_y \times \partial o_y / \partial \ell \\ o_y &= \frac{\exp(\ell_y)}{\sum_i \exp(\ell_i)} \end{aligned}$$

To calculate $\partial o_y / \partial \ell$ we will make use of the derivative rule for quotients:

$$\left(\frac{f(x)}{g(x)} \right)' = \frac{g(x)f(x)' - f(x)g(x)'}{g(x)^2}$$

It is helpful to consider the application of this rule to $\partial o_y / \partial \ell$ in two cases: when $i = k$ and when $i \neq k$. Remember that even though o_y is a scalar, ℓ is a vector, so we're calculating $\partial o_y / \partial \ell_i$ for every member ℓ_i of ℓ .

$$\begin{aligned}
[\partial o_y / \partial \ell]_{i \neq y} &= \frac{\sum_{i'} \exp(\ell_{i'}) \times 0 - \exp(\ell_y) \exp(\ell_i)}{(\sum_{i'} \exp(\ell_{i'}))^2} \\
&= -\frac{\exp(\ell_y)}{\sum_{i'} \exp(\ell_{i'})} \frac{\exp(\ell_i)}{\sum_{i'} \exp(\ell_{i'})} \\
&= -o_y o_i \\
[\partial o_y / \partial \ell]_y &= \frac{\sum_i \exp(\ell_i) \exp(\ell_y) - \exp(\ell_y)^2}{(\sum_{i'} \exp(\ell_{i'}))^2} \\
&= \frac{\exp(\ell_y)}{\sum_{i'} \exp(\ell_{i'})} \frac{\sum_i \exp(\ell_i) - \exp(\ell_y)}{\sum_{i'} \exp(\ell_{i'})} \\
&= o_y(1 - o_y)
\end{aligned}$$

Now we can multiply $\partial L / \partial o_y$ ($-1/o_y$) with $\partial o_y / \partial \ell$ to get $\partial L / \partial \ell$:

$$\partial L / \partial \ell = \begin{cases} o_y - 1 & i = y \\ o_i & \text{otherwise} \end{cases} \quad (6.1)$$

We next continue on down to find the gradient of L with respect to O and b_O , which are actual parameters we want to learn. We use the definition of ℓ in terms of these variables and what we have previously learned:

$$\begin{aligned}
\partial L / \partial O &= \partial L / \partial \ell \times \partial \ell / \partial O \\
\partial L / \partial b_O &= \partial L / \partial \ell \times \partial \ell / \partial b_O \\
\ell &= hO + b_O \\
\partial \ell / \partial O &= h \\
\partial \ell / \partial b_O &= 1
\end{aligned}$$

We can simply multiply $\partial L / \partial \ell$, which is the complicated value in Equation 6.1, by either h or (the vector) 1, as noted above. Here it's worth noting that we want to get the shapes of our gradient matrices right and that we want to deal with *batches* of training samples properly.

Imagine that we are updating parameters after seeing one training instance. Then, $\partial L / \partial \ell$ is a (1×5) vector. h is a (1×1000) vector, and we want to update O , which is a (1000×5) matrix. Thus we take $h^T \times \partial L / \partial \ell$ to get the right shape. However note that in general we do not update after a single training instance; rather there may be some d items in the *minibatch*. So in fact $\partial L / \partial \ell$ is a $(d \times 5)$ matrix and h is a $(d \times 1000)$ matrix. $h^T \times \partial L / \partial \ell$ still yields a (1000×5) matrix but it is actually the sum of d individual loss calculations. The point of batch updating is to take a per-item average. Thus the proper update for O is to subtract (the learning rate times) $\frac{h^T \times \partial L / \partial \ell}{d}$. Similarly, to update b_O , it is important to actually multiply $\partial L / \partial \ell$ by a length-5 ones vector, which amounts to summing each dimension of $\partial L / \partial \ell$ along the batch axis, then divide by d .

If you've gotten this far, the rest should be straight-forward. We will need $\partial L / \partial h$, which is of course $\partial L / \partial \ell \times \partial \ell / \partial h$; the former term is in Equation 6.1 and is $(d \times 5)$, the latter is

simply O , which is (1000×5) . We calculate as $\partial L / \partial \ell \times O^T$ to get a $(d \times 1000)$ result for $\partial L / \partial h$.

We can now move on to the hidden layer; let's assume g is ReLu.

$$\begin{aligned}\partial L / \partial z &= \partial L / \partial h \times \partial h / \partial z \\ h &= \text{ReLU}(z) \\ \partial h / \partial z &= \begin{cases} 1 & z \geq 0 \\ 0 & \text{otherwise} \end{cases}\end{aligned}$$

$$\begin{aligned}\partial L / \partial H &= \partial L / \partial z \times \partial z / \partial H \\ \partial L / \partial b_H &= \partial L / \partial z \times \partial z / \partial b_H \\ z &= xH + b_H \\ \partial z / \partial H &= x \\ \partial z / \partial b_H &= 1\end{aligned}$$

We update H , a $(50,000 \times 1000)$ matrix with $-\partial L / \partial H$; The dimensions of $\partial L / \partial z$ are $(d \times 1000)$, the dimensions of $\partial z / \partial H$ are $(d \times 50,000)$; thus we form $\partial L / \partial H = \frac{(\partial z / \partial H)^T \times \partial L / \partial z}{d}$. Similarly we update b_H , a (1×1000) vector with $-\partial L / \partial b_H$; we multiply $\partial L / \partial z$ by a 1000-length ones vector which sums its values along the d axis, then divide by d .

6.1.2 Word embeddings

Previously we let x , with dimension $|V|$, represent a bag of words and be the input features. This does not allow the relative positions of the words in the input to be specified. A more common approach is to instead use a fixed sequence of some t (let's say 20) words, and represent each word in the vocabulary by an e -dimensional vector. This fits in nicely with our set of equations. Let E be a $|V| \times e$ matrix (often called an *embedding table*). Informally, we assign an index for each word in the vocabulary from 1 to $|V|$. Let the input be j_1, j_2, \dots, j_t where each j_i is a *one-hot vector*, i.e. if j_i represents 'salamander' and the index for that word is 48, then $j_i = 0, \dots, 0, 1, 0, \dots, 0$ consisting of 47 0s, a 1, and then 49,952 0s. Then we redefine x as $j_1E; j_2E; \dots; j_tE$, a te -length vector. Backpropagation is extended to update E as well³.

³There is generally no bias term for the word embeddings

Chapter 7

Language Models: N-gram & Feed-Forward

7.1 What they are and why they're useful

A language model is, formally, a probabilistic formal language, i.e. an n -tuple that includes a vocabulary Σ and contains mapping mechanism $\Sigma^* \rightarrow \mathbb{R}_{\geq 0}$. Furthermore, the sum of all (infinite) $\mathbf{x} = x_1, x_2, \dots, x_n \in \Sigma^*$ should be 1. Practically speaking we usually want to answer the question “What is the probability of the next word?”

We are formally seeking $P(x_1, x_2, \dots, x_n)$ so we can conveniently use the chain rule, insert start and end tokens x_0 , x_{stop} , and re-express as $P(x_1|x_0)P(x_2|x_1, x_0)$

$P(x_3|x_2, x_1, x_0) \dots P(x_{\text{stop}}|x_n, \dots, x_0)$.

Why do we care? This can cover both syntax

$P(\text{the cat slept peacefully}) > P(\text{slept the peacefully cat})$

and semantics

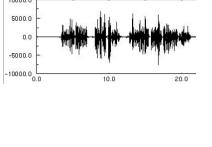
$P(\text{she studies morphosyntax}) > P(\text{she studies more faux syntax})$

Furthermore the notion can be generalized beyond a single sentence and model a continuous stream of language.

Language models help us to generate:

- translations
- spelling/grammar corrections
- summarizations
- text recognized from speech

Task	Input	Options	Final (post-LM)
spelling correction	no much effort	no much effect so much effort no much effort not much effort	not much effort

speech recognition		she studies morphosyntax she studies more faux syntax she's studies morph or syntax	she studies morphosyntax
translation	ella se va a casa	she is going home she is going house she goes to home to home she is going	she is going home

These can also be used for prediction (type ‘Where can I’ into google, or start typing a text message).

This application comes out of the generative models we looked at before. If we wanted originally some $P(Y|X)$ this is equivalent to $P(X|Y)P(Y)/P(X)$ where $P(X|Y)$ can be thought of as a ‘noisy channel’ corrupting unobserved Y into X . Then $P(Y)$ is the language model the data creator used when generating Y before corrupting it into X , which is what is seen. ($P(X)$ isn’t needed; we see X , we don’t care about its likelihood). Y could be anything; it could be a sentiment (but that’s not much of a language) or it could be a tag sequence, or a language sequence. We’ll increasingly consider cases where it’s a natural language sequence.

7.2 N-gram models

Just like we did for POS tags in the HMM, we can make an independence assumption, e.g. that $P(x_i|x_{i-1}, x_{i-2}, x_{i-3}, x_{i-4}) = P(x_i|x_{i-1}, x_{i-2})$ (trigram model). And we can estimate these conditional probabilities from data, just like before.

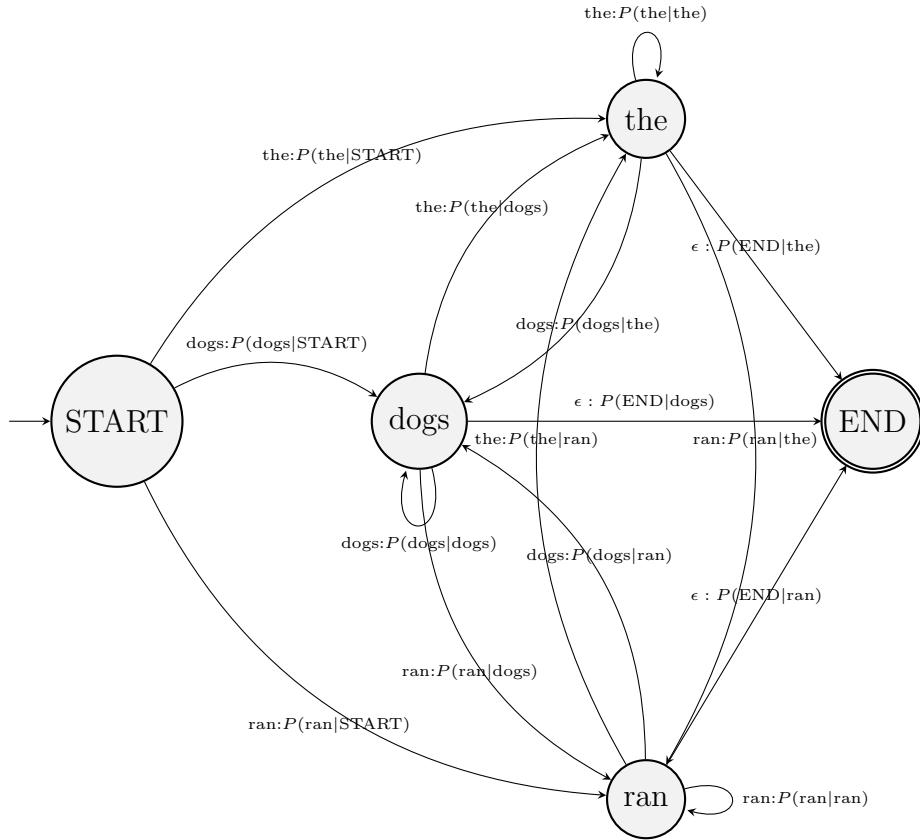
So if you want $P(\text{mast}|\text{before, the})$ we can use a corpus (say, Moby Dick) and unix tools from before:

```
sed 's/ /\n/g' moby dick.txt | grep -v "^\$" > md.words
paste md.words.txt <(tail -n+2 md.words.txt) <(tail -n+3 md.words.txt) \
    | grep -ic '^before\the\t'
29
paste md.words.txt <(tail -n+2 md.words.txt) <(tail -n+3 md.words.txt) \
    | grep -ic '^before\the\tmast'
4
```

So, $4/29 = 13.8\%$ (at least in nautical novels).

7.2.1 N-gram as FSA

Just as we saw with POS tags, FSAs are suitable for representing n-gram language models; in fact this was actually how they were represented (e.g. back in pre-neural speech recognition days). Here is an example of a bigram FSA:



7.2.2 Using N-gram language models

Language models can be used for both *evaluation* and *generation*.

For evaluation, imagine we had the following snippet of text and wanted to know its probability.

`Call me Ishmael . Some years ago never mind how long precisely`

A 3-gram language model would estimate this as $P(\text{Call}|\text{START}, \text{START})P(\text{me}|\text{Call}, \text{START})P(\text{Ishmael}|\text{Call}, \text{me})P(\cdot|\text{me}, \text{Ishmael})\dots$. This could be used when comparing different alternatives, as above.

For generation, we proceed as follows: Let's say you've already started with `Call`. Then from the set of $P(\cdot|\text{Call}, \text{START})$, *sample* a word proportionally to the distribution. E.g. If we have:

x	$P(x \text{call})$
him	.179
it	.143
of	.071
the	.071
me	.071
all	.036
our	.036
...	...

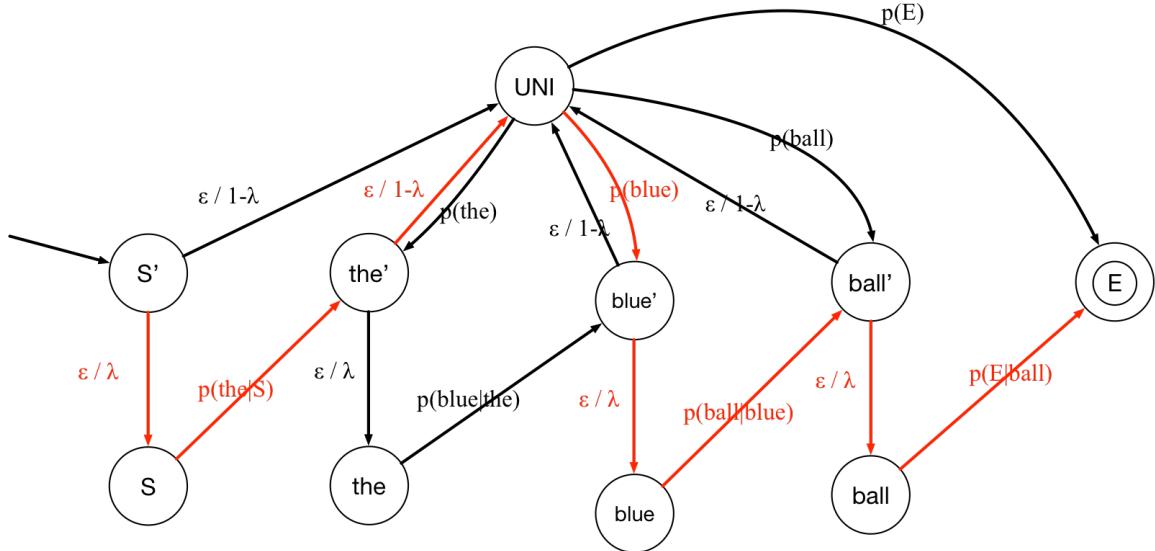
you imagine a wheel where `him` takes up 17.9% of the wheel, `it` the next 14.3% and so on. You spin the wheel and choose the word you land on. Let's say you get `it`. Then you choose from $P(\cdot|it, \text{call})$ using a new table, and so on. Note that it's generally not a good idea to just take the most probable argument, nor is it a good idea to sample uniformly.

7.2.3 Problems with n-grams: sparsity, (Backoff and Smoothing) and storage

N-gram language models are of great utility but they have some problems that need handling. For one thing, they are quite *sparse*. 99.8% of the 5-grams in Moby Dick, for instance, occur exactly once¹.

What to do? One thing we can do is, as before, smooth. So if “go to sea as the” does not occur in training (it doesn’t in Moby Dick) we can still add some small amount to each vocabulary term so we don’t get zero probability for the whole sentence.

But what if “head to sea as a” does not occur because the context, “head to sea as $_$ ” does not occur? Explicitly smoothing every possible 4-gram would explode the memory needed to represent the language model. We instead (or really, additionally) condition on 3-gram context and interpolate between the two models, i.e. $\lambda P(a|4\text{-gram}) + (1-\lambda)P(a|3\text{-gram})$. Even this can be represented as an FSA:



Note another problem with n-gram language models is their size. There is a parameter (a probability) for every n-gram seen in training, plus for some n-grams not seen in training (due to smoothing), plus all $n - k$ grams for $k = 1$ to $n - 1$ (due to backoff). More training data makes for better language models, but also for larger language models. Lossless (e.g. trie storage) and lossy (e.g. Bloom filters – a hash function based approach that was sometimes wrong but with low probability) compression techniques were all the rage until about 2011, but we won’t discuss them here so we can instead move on to neural language models, which made these approaches unnecessary.

¹The three most frequent 5-grams, each occurring four times, are [in the middle of the], [go to sea as a], and [' Queequeg,' said I, ']

7.3 Intrinsic evaluation: Perplexity

There's no way to conceive of held-out 'labeled' data in language modeling. So how are we to judge the quality of a language model? We hold out some portion of natural language and after building the model ask it what it thinks of the held-out portion (i.e. how probabilistic it is). Since the held-out portion is a sample of real language, the model should give it a high probability. Naturally, we wouldn't expect the probability to be 1, as if the model is a true language model, it should distributed probability mass across all (generally infinite) sentences of the language.

It won't do, though, to have models report probabilities on one particular piece of text (as there will be overfitting) nor can we compare across different pieces of text, as they have different sizes. We instead want to report *per-word* behavior, since although we know words don't have equal 'amounts' of language (whatever that means) they come as close as anything else.² Rather than simply describe the probability per word we use an information-theoretical approach. Consider *cross-entropy*, which is used to train logistic regression and neural models (particularly ones with categorical output, which an LM is an instance of). The cross entropy being calculated is:

$$H(\tilde{p}, q) = - \sum_{x \in \mathcal{X}} \tilde{p}(x) \log q(x)$$

where x is a possible member of language X (i.e. a sentence), \tilde{p} is the 'true' distribution of language, and q is our model's distribution of language. H , the cross-entropy, measures the average number of bits (assuming a log base of 2) it takes to properly calculate the members of X using the suboptimal model q instead of \tilde{p} . Think of this as a 'codebook' with instructions on how to turn faulty distribution q into the true distribution; H is the size of the entry of each item in the book. We don't know the true distribution of \tilde{p} , but we have a sample M of it, so we assume every $x \in M$ has probability 1, and everything else has probability 0. So we can rewrite as:

$$H(\tilde{p}, q) = - \sum_{x \in \mathcal{M}} \log q(x)$$

Furthermore, we generally predict language one word at a time and we want a metric for the average 'goodness' of our model per word. So we re-cast M as a sequence of words $x_1, x_2, \dots, x_{|M|}$ and write the average cross-entropy as:

$$H_{\text{ave}}(\tilde{p}, q) = - \frac{1}{|M|} \sum_{x_i \in \mathcal{M}} \log q(x_i | x_{i-1}, \dots, x_1)$$

Rather than report the average number of bits needed to represent the truth using q instead of \tilde{p} we instead cast this as the 'degree of confusion.' If 5 bits were needed, then a codebook with words that represent up to $2^5 = 32$ choices are needed, and so we are 32-way 'confused' or 'perplexed' when we use q . We thus calculate the *per-word perplexity* as

$$2^{H_{\text{ave}}(\tilde{p}, q)}$$

²There is work that measures per-character or even per-byte perplexity but it is less common.

However, this assumes cross-entropy was calculated with a base 2 log, and in general that's not what we do; we prefer to use a base of e . So perplexity is in fact usually calculated in terms of 'nats', i.e.

$$\exp(H_{\text{ave}}(\tilde{p}, q))$$

A really bad language model that predicts each word in V uniformly would get perplexity of $|V|$. A really good one would have perplexity of 1.

Despite using per-word averages it is usually a good idea to compare across common benchmarks. The 1m-word Penn treebank with a vocabulary limited to 10,000 types gets 141 ppl using a good 5-gram model, and under 60 using neural models (see below/next). On a larger wikipedia-based 1b-word corpus, state of the art as of 2016 is around 25.

7.4 Feed-forward language models

TG previously introduced feed forward language models for classification. How can we use these for language modeling? Further, why might these be useful for language modeling?

First the how. Although this can be multi-layered, let's use a simple one-layer architecture, assume a context of four words, and a non-linear function for the hidden layer called g . We assume the dimension of a word representation (aka the 'embedding') is 50 and the dimension of the hidden representation is 100. We also assume a vocabulary of 20,000 words. When calculating the hidden and output vectors (but not the embedding) we will assume a bias term. We thus have the following weight (i.e. parameter) matrices, which are listed along with their dimensions:

- Embedding table E : (20,000 x 50)
- Hidden weights H : (200 x 100)
- Hidden bias b_H : (1 x 100)
- Output weights O : (100 x 20,000)
- Output bias b_O : (1 x 20,000)

E is indexed by the vocabulary. We want to get the probability of the example above, in a 5-gram model. So that is $P(\text{Call}|\text{START}, \text{START}, \text{START}, \text{START})$

$P(\text{me}|\text{Call}, \text{START}, \text{START}, \text{START})P(\text{Ishmael}|\text{me}, \text{Call}, \text{START}, \text{START})$, etc. For the first term, $P(\text{Call}|\text{START}, \text{START}, \text{START}, \text{START})$, assume we have some row in E for START. Concatenate four copies of that row; we'll call that x . Then multiply and apply nonlinear function g : $g(xH + b_H) = h$; a 100-dimensional vector representing the context. We want to determine the probability of Call given this context. $hO + b_O$ yields ℓ , a 20,000-dimensional vector with one value for each word in the vocabulary, including Call. But these are not probabilities³. We use the softmax operator to set $o_i = \frac{\exp(\ell_i)}{\sum_{i'} \exp(\ell_{i'})}$ for each position i in ℓ . Then if we assume position k in o (and ℓ) corresponds to the word Call, we simply retrieve that value to get $P(\text{Call}|\text{START}, \text{START}, \text{START}, \text{START})$. We next get

³we call ℓ *logits*

$P(\text{me}|\text{Call}, \text{START}, \text{START}, \text{START})$ by doing the same thing, but starting off with the k th row of E concatenated with three copies of the row for START, and at the end choose the item corresponding to the index for me.

What is the advantage to using this approach instead of the non-neural n -gram approach (apart from the empirical superior performance)?

Smoothing/Interpolation: Notice that for any 4-gram over the vocabulary space we can get the probability of each word (also over the entire vocabulary space). There is no explicit backoff or smoothing here.

Storage: Let's compare the sizes of the feedforward and non-feedforward models. First sum up the feedforward:

- E : 1,000,000
- $H + b_H$: 20,100
- $O + b_O$: 2,020,000
- Total: 3,040,000

Notice that this is not dependent on how much training data we use. It is dependent on some modeling decisions, like embedding and hidden size, number of layers, etc. Most of all it's dependent on vocabulary. Contrast this with the number of parameters used for a 5-gram model. This is highly dependent on corpus size. Below we show the parameter size if the model is trained on moby dick and tom sawyer and compare this to training on the treebank portion of the wall street journal. Note that both are considered fairly small corpora for language modeling. I also added stats for gigaword, a much larger corpus.

n-gram	moby dick + tom sawyer	wsj treebank	gigaword
(tokens)	181,017	1,107,391	4.2b
5	180,612	1,074,244	558m
4	178,904	1,037,648	447m
3	167,372	906,848	246m
2	113,439	530,884	60m
1	27,279	85,967	1m
total	667,606	3,635,591	1.3B

The feedforward model, meanwhile, remains a constant size.

7.4.1 Why should this work?

To some degree this is a bit of a mystery on a deep level. But there is some good intuition for why this might work. There are multiple angles to address this question; here's one:

A word embedding can be thought of as features specific to that word type. Rather than treat each of 20k words as 20k independent items, we characterize them as having certain properties. We could do this by hand (e.g. 'French origin', 'animate', 'plural') but rely on training for these features to be implicitly defined. By representing a 20,000-dimensional object in 50-space, some properties shared among disparate objects have to be identified, otherwise learning won't happen.

Similarly, the hidden representation is a 100-dimensional representation that generalizes a four-gram – there are $20,000^4$ possible such objects. This is formed by considering multiple embedding features together simultaneously. If we added more hidden layers this would consider multiples of multiples of features simultaneously.

7.4.2 How are these parameters set?

With backpropagation over a training corpus, as TG showed. Let's go over this in a little more detail, especially because of the relevance to HW2. We'll build a computation graph and discuss how to calculate the gradient updates at each position, how to deal with batches of training, etc.

A training instance consists of 4-gram context and the next word. Note that we have declared our vocabulary to be 20,000 words; even for Treebank-wsj that is too small a vocabulary to cover all types seen.⁴ Vocabulary is the biggest factor in network size, so computational power generally limits it; if we assume 20k is the cap here, we will have to replace some selection of word types (typically some 1-count types).⁵

The loss is typically the cross-entropy, $-\sum_{i \in V} \tilde{p}(w_i|c) \log o_i$ for context (i.e. 4-gram) c , where i is an index to a word in vocabulary V (which is represented as w_i ; $\tilde{p}(w_i|c)$ means the ‘true’ probability of word w_i in this context, and o_i , as previously mentioned, is the model’s probability of the training example.

7.4.3 Getting the gradients right

As we noted before:

$$\begin{aligned} z &= xH + b_H \\ h &= g(z) \\ \ell &= hO + b_O \\ o &= \text{softmax}(\ell) \\ L &= -\log(o_k) \end{aligned}$$

where L is the loss; note that the variable on the right is o_k , the probability of word w_k , which is the ‘true’ word for this context. Having calculated the loss, we update each set of parameters (H, b_H, O, b_O) with respect to the opposite of the gradient of L , i.e.:

$$\begin{aligned} H &\leftarrow H - \lambda \partial L / \partial H \\ b_H &\leftarrow b_H - \lambda \partial L / \partial b_H \\ O &\leftarrow O - \lambda \partial L / \partial O \\ b_O &\leftarrow b_O - \lambda \partial L / \partial b_O \end{aligned}$$

⁴English dictionaries have about 170k entries; this doesn’t cover all inflections but gives you an idea of how many words might be ‘enough.’

⁵Later on we’ll consider using word pieces which directly addresses this vocabulary problem.

where λ is a learning rate. Now how are these partials determined? We start at the loss equation itself and use simple calculus:

$$L = -\log(o_k)$$

$$\partial L / \partial o_k = -1/o_k$$

Now consider the definition of o_k itself; we can use the chain rule and the local derivative of o_k with respect to ℓ , though softmax is a slightly tricky function to take a derivative of:

$$\partial L / \partial \ell = \partial L / \partial o_k \times \partial o_k / \partial \ell$$

$$o_k = \frac{\exp(\ell_k)}{\sum_i \exp(\ell_i)}$$

To calculate $\partial o_k / \partial \ell$ we will make use of the derivative rule for quotients:

$$\left(\frac{f(x)}{g(x)} \right)' = \frac{g(x)f(x)' - f(x)g(x)'}{g(x)^2}$$

It is helpful to consider the application of this rule to $\partial o_k / \partial \ell$ in two cases: when $i = k$ and when $i \neq k$.

$$\begin{aligned} [\partial o_k / \partial \ell]_{i \neq k} &= \frac{\sum_{i'} \exp(\ell_{i'}) \times 0 - \exp(\ell_k) \exp(\ell_i)}{(\sum_{i'} \exp(\ell_{i'}))^2} \\ &= -\frac{\exp(\ell_k)}{\sum_{i'} \exp(\ell_{i'})} \frac{\exp(\ell_i)}{\sum_{i'} \exp(\ell_{i'})} \\ &= -o_k o_i \\ [\partial o_k / \partial \ell]_k &= \frac{\sum_i \exp(\ell_i) \exp(\ell_k) - \exp(\ell_k)^2}{(\sum_{i'} \exp(\ell_{i'}))^2} \\ &= \frac{\exp(\ell_k)}{\sum_{i'} \exp(\ell_{i'})} \frac{\sum_i \exp(\ell_i) - \exp(\ell_k)}{\sum_{i'} \exp(\ell_{i'})} \\ &= o_k (1 - o_k) \end{aligned}$$

Now we can multiply $\partial L / \partial o_k$ ($-1/o_k$) with $\partial o_k / \partial \ell$ to get $\partial L / \partial \ell$:

$$\partial L / \partial \ell = \begin{cases} o_k - 1 & i = k \\ o_i & \text{otherwise} \end{cases} \quad (7.1)$$

We next continue on down to find the gradient of L with respect to O and b_O , which are actual parameters we want to learn. We use the definition of ℓ in terms of these variables and what we have previously learned:

$$\begin{aligned} \partial L / \partial O &= \partial L / \partial \ell \times \partial \ell / \partial O \\ \partial L / \partial b_O &= \partial L / \partial \ell \times \partial \ell / \partial b_O \\ \ell &= hO + b_O \\ \partial \ell / \partial O &= h \\ \partial \ell / \partial b_O &= 1 \end{aligned}$$

We can simply multiply $\partial L / \partial \ell$, which is the complicated value in Figure 7.1, by either h or (the vector) 1, as noted above. Here it's worth noting that we want to get the shapes of our gradient matrices right and that we want to deal with *batches* of training samples properly.

Imagine that we are updating parameters after seeing one training instance. Then, $\partial L / \partial \ell$ is a (1 x 20,000) vector. h is a (1 x 100) vector, and we want to update O , which is a (100 x 20,000) matrix. Thus we take $h^T \times \partial L / \partial \ell$ to get the right shape. However note that in general we do not update after a single training instance; rather there may be some d items in the *minibatch*. So in fact $\partial L / \partial \ell$ is a (d x 20,000) matrix and h is a (d x 100) matrix. $h^T \times \partial L / \partial \ell$ still yields a (100 x 20,000) matrix but it is actually the sum of d individual loss calculations. The point of batch updating is to take a per-item average. Thus the proper update for O is to subtract (the learning rate times) $\frac{h^T \times \partial L / \partial \ell}{d}$. Similarly, to update b_O , it is important to actually multiply $\partial L / \partial \ell$ by a 20,000-length ones vector, which amounts to summing each dimension of $\partial L / \partial \ell$ along the batch axis, then divide by d .

If you've gotten this far, the rest should be straight-forward. We will need $\partial L / \partial h$, which is of course $\partial L / \partial \ell \times \partial \ell / \partial h$; the former term is in Equation 7.1 and is (d x 20,000), the latter is simply O , which is (100 x 20,000). We calculate as $\partial L / \partial \ell \times O^T$ to get a (d x 100) result for $\partial L / \partial h$.

We can now move on to the hidden layer; let's assume g is ReLu.

$$\begin{aligned}\partial L / \partial z &= \partial L / \partial h \times \partial h / \partial z \\ h &= \text{ReLU}(z) \\ \partial h / \partial z &= \begin{cases} 1 & z \geq 0 \\ 0 & \text{otherwise} \end{cases}\end{aligned}$$

$$\begin{aligned}\partial L / \partial H &= \partial L / \partial z \times \partial z / \partial H \\ \partial L / \partial b_H &= \partial L / \partial z \times \partial z / \partial b_H \\ z &= xH + b_H \\ \partial z / \partial H &= x \\ \partial z / \partial b_H &= 1\end{aligned}$$

We update H , a (200 x 100) matrix with $-\partial L / \partial H$; The dimensions of $\partial L / \partial z$ are (d x 100), the dimensions of $\partial z / \partial H$ are (d x 200); thus we form $\partial L / \partial H = \frac{(\partial z / \partial H)^T \times \partial L / \partial z}{d}$. Similarly we update b_H , a (1 x 100) vector with $-\partial L / \partial b_H$; we multiply $\partial L / \partial z$ by a 100-length ones vector which sums its values along the d axis, then divide by d .

We didn't go back all the way to E but we could have; $\partial L / \partial E = \partial L / \partial x \times \partial x / \partial E$. To get $\partial x / \partial E$ note that x is formed by concatenating four rows of E ; let the selector of those rows be (d x 20,000) one-hot (per row) matrix Φ . So consider this four separate instances of $x = \Phi E$. Take the (d x 200) matrix $\partial L / \partial x$ and slice it into columns 0-49, 51-99, 100-149, 150-199. Then use each (d x 50) matrix with $\partial x / \partial E = \Phi$; the net result is that the selected rows are updated by the corresponding slice of $\partial L / \partial x$.

Chapter 8

Language Models Continued: RNNs

8.1 Limitations of Feed-Forward Networks

Feed-forward language models solve a lot of the problems with (non-neural) n-gram models. Specifically they

- generalize, handling the sparsity problem much better than n-gram models, with much lower perplexity on unseen n-grams.
- are efficient, requiring constant memory and do not blow up with the amount of training data (and hence novel n-grams) seen.
- allow for larger n ; beyond 5-grams, non-neural models were never that helpful. We used 12-gram feed forward models effectively.

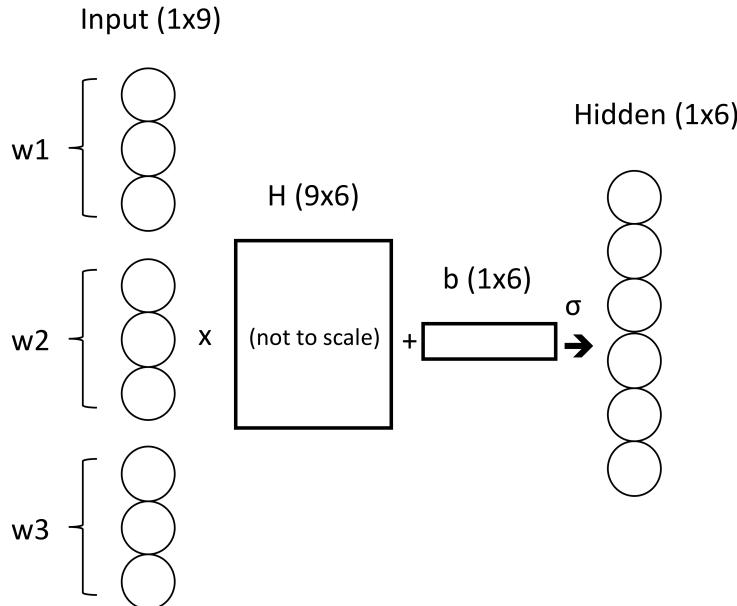
There are some limitations, however. Some are newly introduced, while some are persistent and may now be addressed.

- Although the size of feed-forward is fixed, it is highly dependent on vocabulary size (this also determines computation time). This limits the vocabulary rather significantly. There have been methods to overcome this, either partially or totally, some of which we will talk about (subword models) and some of which we won't (noise-contrastive estimation, hierarchical softmax).
- Compared to non-neural models, neural models take a rather long time to train, since parameter estimation using maximum likelihood and smoothing requires only one pass through the data, (with very simple calculations). The use of GPUs helps this somewhat but it remains an ongoing concern we won't directly discuss.
- Correctly modeling language can require very long context. E.g. *My mother, who once caught a balance beam in her eye, but is actually nicer than you might expect, lived with three bermuda sharks and (like/likes) ice cream..* We are fundamentally limited by whatever n we select. We will directly address this now.

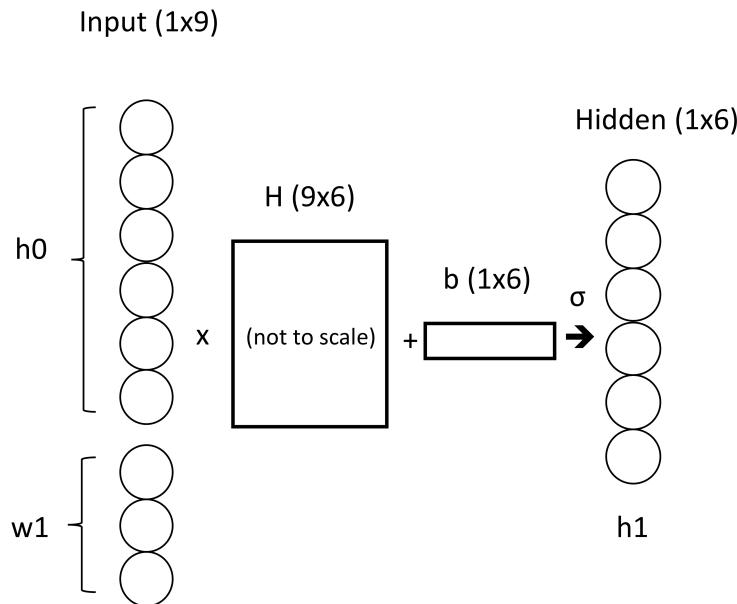
8.2 Recurrent Neural Networks (RNNs)

In some respects, RNNs are very similar to feed-forward networks. A word representation feeds into one or more hidden layers, fully connected and with a nonlinear function, and ultimately the hidden representation is used to predict the probability of the next word. Gradient descent along the cross-entropy loss via backpropagation is used to update parameters.

The key difference is in the structure of the parameters, specifically in the construction of the hidden vector, h . Here is the way h is constructed in feed-forward, as a 4-gram model with an embedding dimension of 3 and a hidden dimension of 6:

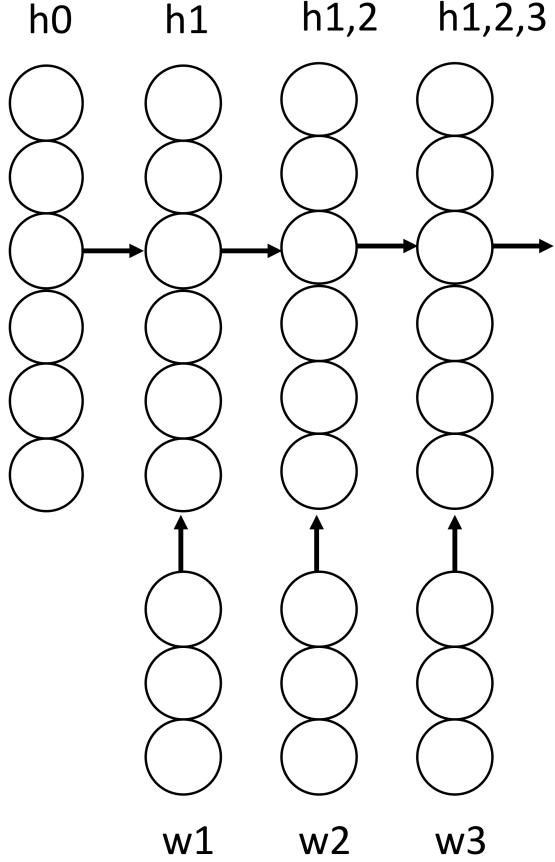


By comparison here is the construction of a hidden unit h_1 for an RNN:



Notice that the input to the RNN is a hidden vector and a word embedding. The output

is another hidden vector.¹ However, this only captures the hidden representation for one word. If we want to capture the sequence w_1, w_2, w_3 we simply re-do the calculation, using the last calculated h . Note that at each step we use the *same* H and b_H and these are not shown in the diagram below for space reasons.

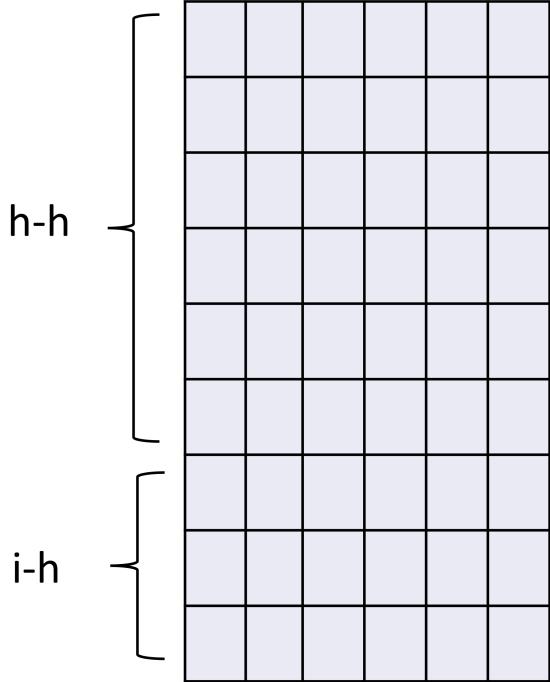


$h_{1,2,3}$ is a representation of w_1, w_2, w_3 the same as the h in the feed-forward example. Just as in the feed-forward case, we can get a logit layer from $h_{1,2,3}$ (which we will now call simply h_3) and then with softmax get probabilities over the next word. But of course we can continue adding words and getting new hidden vectors that capture more and more context.

The hidden vectors can be viewed as (and I often describe them as) *states* in the sense of a finite-state automaton. Each position in hidden space is a state, from which an arc labeled with each word in the vocabulary leads to another state. Different from an FSA, however, is that an RNN is an *infinite*-state automaton (at least to the level of representability in hardware).

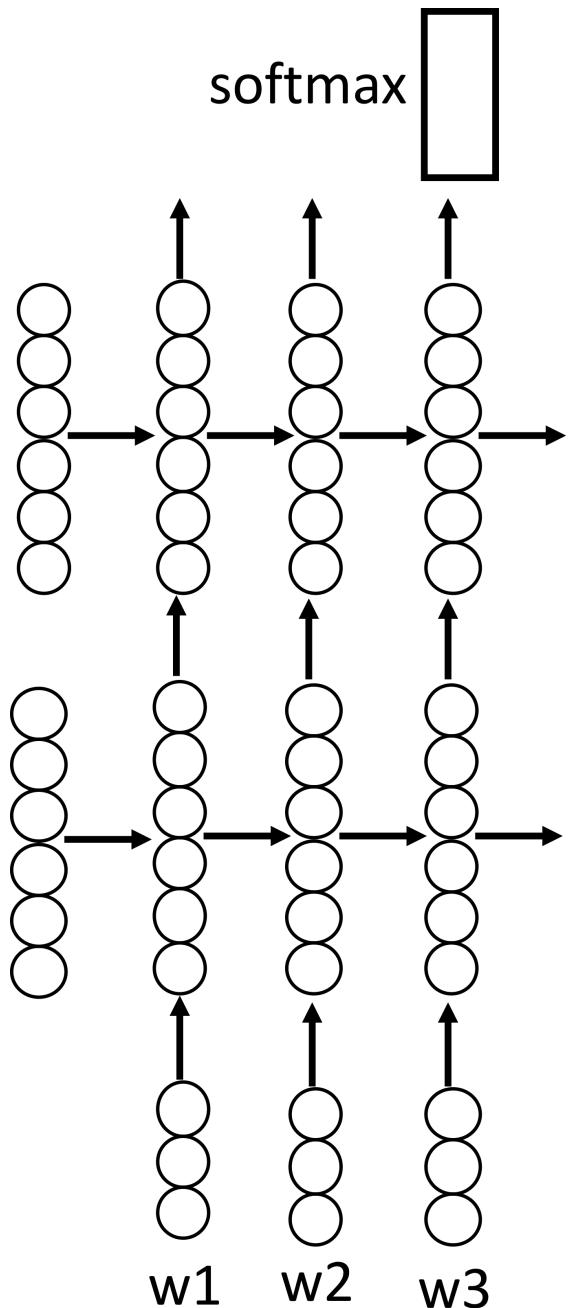
Note that H can be divided into the piece that applies to the hidden state input and the piece that applies to the lexical input; the pieces are frequently written as H_{ih} and H_{hh} to distinguish between the weights relevant to the context (hidden-to-hidden) and those relevant to the words (input-to-hidden):

¹There's no special requirement about the size of the hidden or embedding vectors; I just made them look very similar to illustrate how similar the computation is.



Recall that feed-forward parameters were dependent on the amount of context that was used; for each additional word of context there were V hidden more parameters; as V can be large this is quite substantial.

As with feed-forward networks, RNNs can be *stacked*; this is where the ‘deep’ in deep learning comes from. The hidden unit at a layer becomes the ‘input’ to the next layer. Typically, the last layer is then converted (via output weights) to logits to predict the next word in the sequence. Typically, each layer has a separate learned h_0 , H , and b_H , which are learned, as is the output weight matrix O and bias b_O . So if the non-linear function is σ , the embedding of word w_j is x_j , the hidden weights at layer i are H_i , $b_{H,i}$ and the initial hidden state $h_{0,i}$, the logits for predicting w_2 would be calculated as $\sigma([\sigma([x_1; h_{0,1}]H_1 + b_{H,1}); h_{0,2}]H_2 + b_{H,2})O + b_O$.



8.2.1 Training

The structure of the training for RNN LMs is quite simple; you constantly evaluate the prediction of the next word given the last context. In theory this is done over the entire corpus, but in practice this can be very slow, since gradients are calculated with respect to each step in the context, so context is limited to the sentence. Standard cross-entropy loss is generally used.

8.3 Unreasonable effectiveness of

it turns out RNNs are pretty powerful generation models; this was not always true of the strictly n-gram models. Andrej Karpathy, in a blog post titled ‘The Unreasonable Effectiveness of RNNs’ built an RNN that operated one *character* at a time and trained it on various kinds of text, then generated samples from the trained models and got surprisingly good outputs.

8.3.1 Shakespeare

PANDARUS:

Alas, I think he shall be come approached and the day When little strain would be attain'd into being never fed, And who is but a chain and subjects of his death, I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul, Breaking and strongly should be buried, when I perish The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord: They would be ruled after this chamber, and my fair nues begun out of the fact, to be conveyed, Whose noble souls I'll have the heart of the wars.

Clown: Come, sir, I will make did behold your worship.

VIOLA: I'll drink it.

8.3.2 Wikipedia

== ’’Declaration of Protectance from Iceland’’ ==

In the late 1970s, [[Deep Seols]] and the Australian Federal Navy in order to establish a police duty of a several federal government of the world.

Since 2004 the state regarded as a [[Suffolk Act 1994]], but the [[Army personality|Armed Forces]] appeared in Paris, despots with Nelson concentrated on) was inaugurated as his father. Heraldry put an attempt to get influent territory register. Hayling among their lost operations, a population of Deliberate countries arrived and Harry Elser, established [[The West Virasan Socialist Wars]] for 16 year and modern democratic 30 [[Justice|booms]] elections to the CDC.

8.3.3 L^AT_EX(some compile bugs had to be fixed)

For $\bigoplus_{n=1,\dots,m} \mathcal{L}_{m,n} = 0$, hence we can find a closed subset \mathcal{H} in \mathcal{H} and any sets \mathcal{F} on X , U is a closed immersion of S , then $U \rightarrow T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \text{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \rightarrow V$. Consider the maps M along the set of points Sch_{fppf} and $U \rightarrow U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ???. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\text{Spec}(R') \rightarrow S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S . We claim that $\mathcal{O}_{X,s}$ is a scheme where $x, x' \in S'$ such that $\mathcal{O}_{X,x'} \rightarrow \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\text{GL}_{S'}(x'/S')$ and we win. \square

To prove study we see that $\mathcal{F}|_U$ is a covering of X' , and \mathcal{T}_i is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and \mathcal{F}_p exists and let \mathcal{F}_i be a presheaf of \mathcal{O}_X -modules on C as a \mathcal{F} -module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{\mathcal{M}}^\bullet = \mathcal{I}^\bullet \otimes_{\text{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1} \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (Sch/S)_{fppf}^{\text{opp}}, (Sch/S)_{fppf}$$

8.4 Other Uses

- Tagging (POS, NE); generate a label at every word
- Sentence classification: e.g. sentiment. Take some hidden state as the 'sentence' state – could be the last, could be an average, then predict class label.
- As a sentence representation in something more complicated (e.g. question answering)
- Generation given context (e.g. speech recognition, machine translation)

In many of the whole-sentence variants above a *bidirectional RNN* is used; I'll give a quick sketch but we'll see this more in MT and IE lectures.

8.5 Variants

As we have seen, RNNs can in theory represent infinite context. In practice it is hard to do this due to some practical considerations. Calculating gradient through time requires repeated multiplications of the hidden weight matrix. It turns out that if the largest eigenvalue of $H < 1$ then the gradient will shrink exponentially (vanish), which means after not many words of context you won't see any effect. Also, if the largest eigenvalue of $H > 1$ then the gradient will grow exponentially (explode) which can cause updates to be too large or even NaN.

Exploding gradients can be clipped: define a maximum gradient amount (I think '5' is often used) and if the L_2 norm exceeds that amount, divide the gradients by $\max/\|g\|$. For

vanishing gradients scaling can be used, but another solution was found: a more complicated RNN that has a *memory* element and a means of learning how much memory to keep from step to step.

8.5.1 Long Short-Term Memory (LSTM)

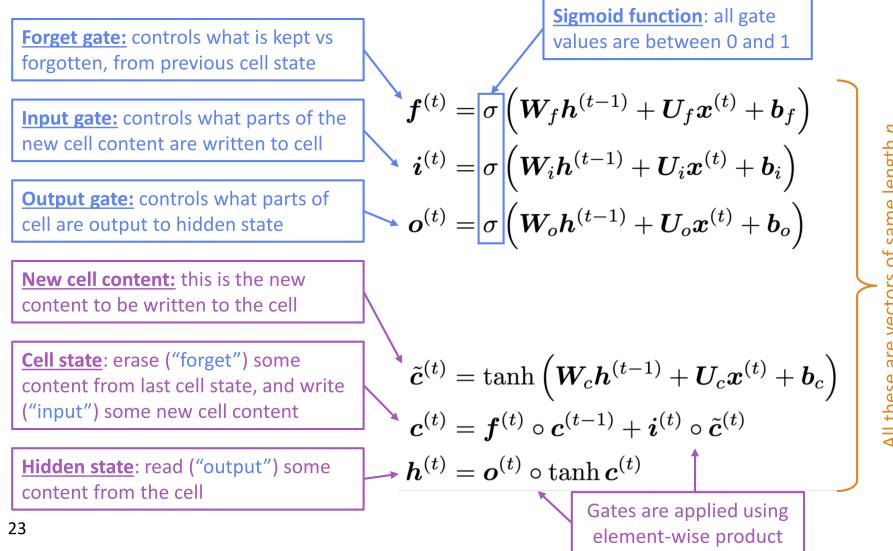
The key to LSTM is as follows:

- The ‘pre-cell’ is a standard nonlinear calculation (typically tanh or Relu).
- Input, forget, and output ‘gates’ are vectors of values from 0 to 1 (typically sigmoid)
- The cell is formed by gating how much should be input from the pre-cell and how much should be forgotten from the last cell (then adding these).
- The hidden state is formed by tanh-ing the cell and then using the output gate to determine how much gets through.

The equation slide from Abi See (stanford class) and figure from Chris Olah (also used by Abi See) help me understand.

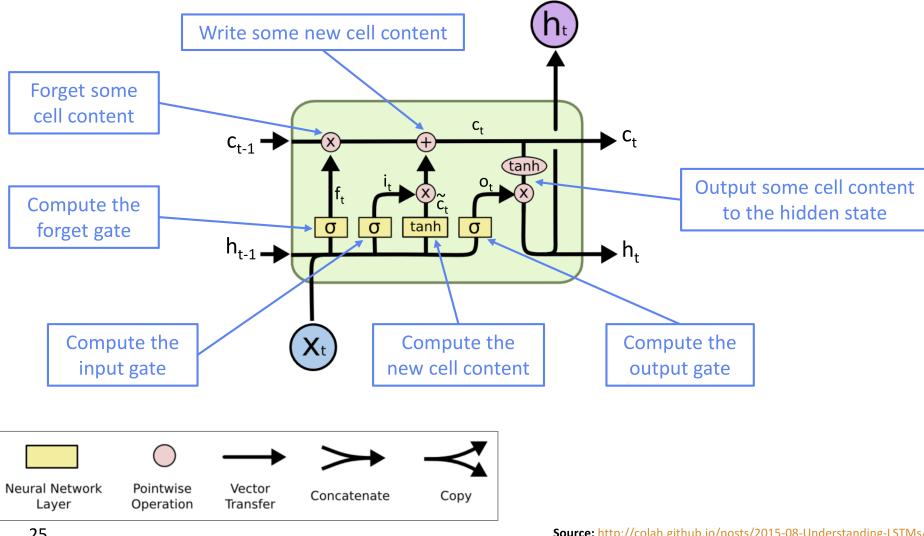
Long Short-Term Memory (LSTM)

We have a sequence of inputs $\mathbf{x}^{(t)}$, and we will compute a sequence of hidden states $\mathbf{h}^{(t)}$ and cell states $\mathbf{c}^{(t)}$. On timestep t :



Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



8.5.2 Gated Recurrent Units

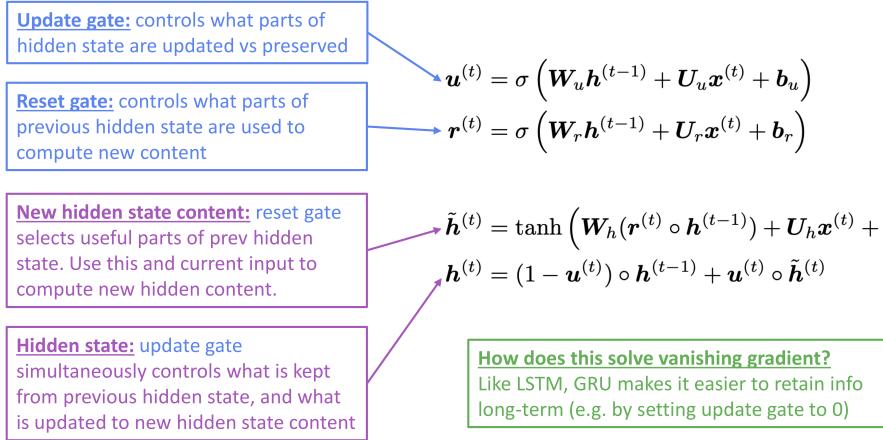
Very similar idea to LSTMs; proposed by Cho et al. in 2014. Simpler than LSTM but same idea:

- No cell state. Two gates; update and reset.
- pre-hidden state calculated like a classic hidden state but hadamard of reset with the previous hidden state (like a forget gate)
- final hidden state interpolates between last hidden and current pre-hidden using update and 1-update (which thus functions like an input and output/forget)

Again, the slides:

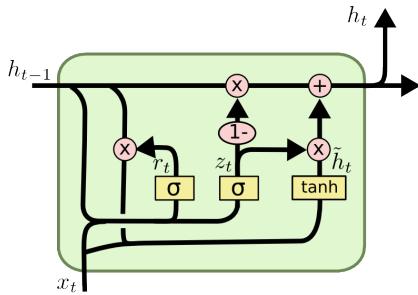
Gated Recurrent Units (GRU)

- Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
- On each timestep t we have input $x^{(t)}$ and hidden state $h^{(t)}$ (no cell state).



28

"Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", Cho et al. 2014, <https://arxiv.org/pdf/1406.1078v3.pdf>



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

There are other variants but LSTM and GRU are the most widely used RNNs. It's unclear which is better; I have mostly used LSTMs but only out of other ways to prevent forgetting the past are the use of *attention* and *residual connections*, which are both ways of 'skipping' the chain of words in one way or another. As we will see, since 2017 (things move fast) Transformer networks have gone a long way to replacing RNNs but use a lot of the techniques that RNNs incorporated to solve these problems.

8.6 LM Summary

- Language Models: predict the next word given previous context.
- statistical n-grams: simple to build, can be tricky to smooth, used for many years
- feed-forward n-grams: avoid sparsity and are more space efficient but longer to train, vocabulary limited, size grows with context
- RNN: infinite context possible. Training takes longer, raises questions of vanishing/-exploding gradients. More complicated models (LSTM, GRU) can help with this.

Chapter 9

Machine Translation

Chapter 10

Transformers

In 2017 some researchers at Google considered whether the recurrent part of RNNs/LSTMs was really that important at all in neural MT. In the paper ‘Attention is all you need’, they described their model, Transformer [30], which outperformed the state of the art at the time at a variety of data points and at lower cost to train.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	3.3 · 10¹⁸	
Transformer (big)	28.4	41.8		$2.3 \cdot 10^{19}$

Within a year or so Transformer models took over most of NLP as they were shown to be useful as language models and as feature sets for classification and structured prediction models. As I write this it’s unclear if yet another model will prove even more compelling but these models seem quite good for now. All the images in these notes come from others’ papers, lectures, blog posts, etc. Apart from the original transformer paper I recommend the illustrated transformer¹ or the annotated transformer².

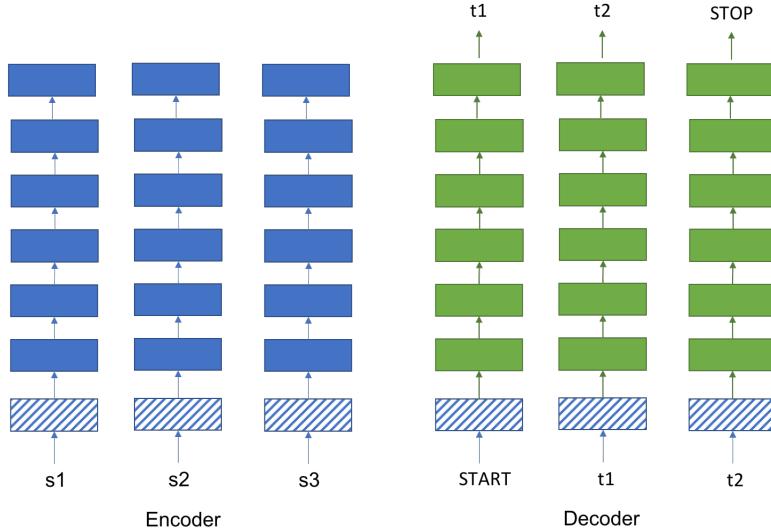
10.1 Base Model

We’re going to cover the details in Transformer in various order, sort of from the outside in. To begin with, the overall shape is stacks of representations, conventionally of size 6, with one representation stack per word in the input and in the output. To begin with let’s imagine each block is just a feed-forward network. Each word is embedded, then at each stage, it’s passed through nonlinear transformation via ReLU. In fact there are two linear transformations and one ReLU at each level. So if x is the embedding (or input from last layer), the output is $\max(0, xW_1 + b_1)W_2 + b_2$.³

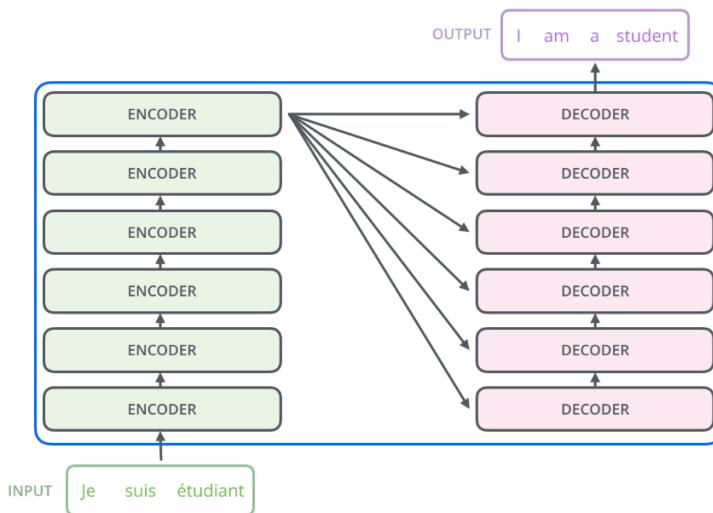
¹<http://jalammar.github.io/illustrated-transformer/>

²<https://nlp.seas.harvard.edu/2018/04/03/attention.html>

³ W_1 is (512×2048) and W_2 is (2048×512) .



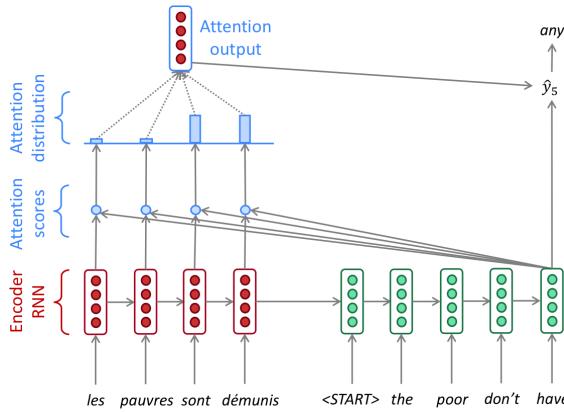
And as you might imagine, attention is heavily involved. There are multiple kinds of attention but to begin with let's consider the 'normal' attention we've already discussed, i.e. from source to target words (from Illustrated transformer):



Notice that this attention is performed at *every layer* of the decoder.

10.1.1 Key, Query, Value Attention

Here's how we did attention before (fig from abi see):



let h_1, \dots, h_N be hidden states of the encoder and s_t be the hidden state of the decoder. Then score vector $e^{(t)} = [s_t^T h_1, s_t^T h_2, \dots, s_t^T h_N]$. Then distribution vector $\alpha^{(t)} = \text{softmax}(e^{(t)})$. Then make a linear combination of h ; $a_t = \sum_{i=1}^N \alpha_i^{(t)} h_i$. That is then concatenated to s_t and used to predict.

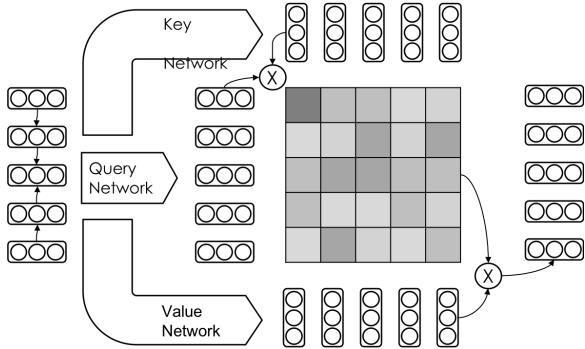
Transformer does it a bit differently. Instead of directly taking a dot product of s_t and each h_i , each of these is transformed linearly; s_t by a “query” matrix Q and h_i by a “key” matrix K . Then $s_t Q(h_i K)^T$ is the score; this is done for every h_i and turned into a distribution α_t by softmax.⁴

Now, instead of using α_t to linearly combine each h_i , the h_i are transformed again by a “value” matrix V . These are then linearly combined. That is then fed to the feed-forward unit. Attention, followed by feed forward, is one layer, and there are six.

10.1.2 self-attention

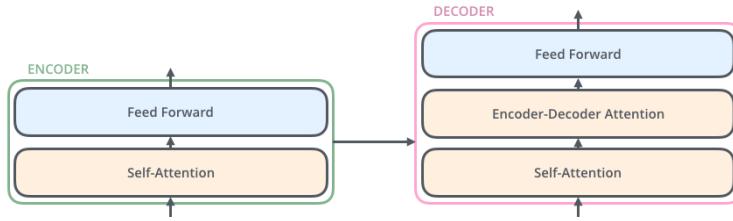
Why should attention be limited to target words looking at related source words? For h_t we can calculate $\alpha_{\text{self}}^{(t)} = h_t Q(h_t K)^T$ on the source side. On the target side we can almost do the same thing; we calculate $s_t Q(s_t K)^T$ but only for $i < t$; otherwise we’d be training on the future, which is not helpful at inference time!. In practice a mask is used to prevent ‘peeking’ on the decoder during training.

I think the figure below by Alireza Zareian nicely expresses the calculation for self-attention:



⁴Not quite. Actually it's $\frac{s_t Q(h_i K)^T}{\sqrt{|K|}}$, i.e. divide by the dimension of K . This keeps gradients from getting too small, per notes in the paper.

So each layer constitutes a number of *sublayers*. Jay Alammar of Illustrated Transformer has a nice figure:

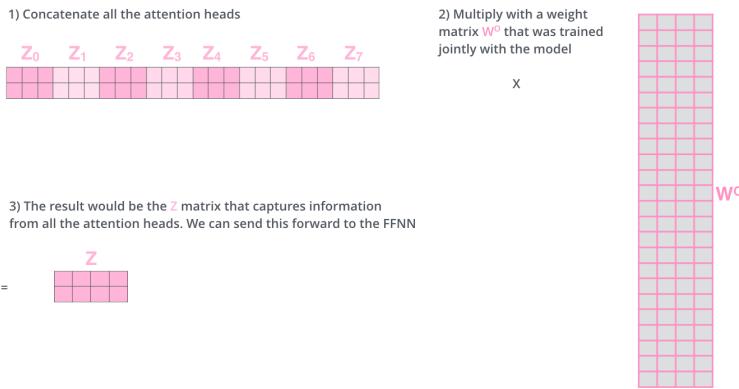


10.1.3 multi-head

Self attention can be viewed as a generalization of convolving kernels used in convolutional neural networks (CNNs). CNN filters, however, have dimension tied to the relative offset of adjacent inputs (words, pixels, hidden units) while the same Q, K, and V are applied to each input on a layer (different set for source, self-encoder, and self-decoder). Also, CNN filters do fixed combination, not a distributional interpolation. But the information sharing paradigm is very similar.

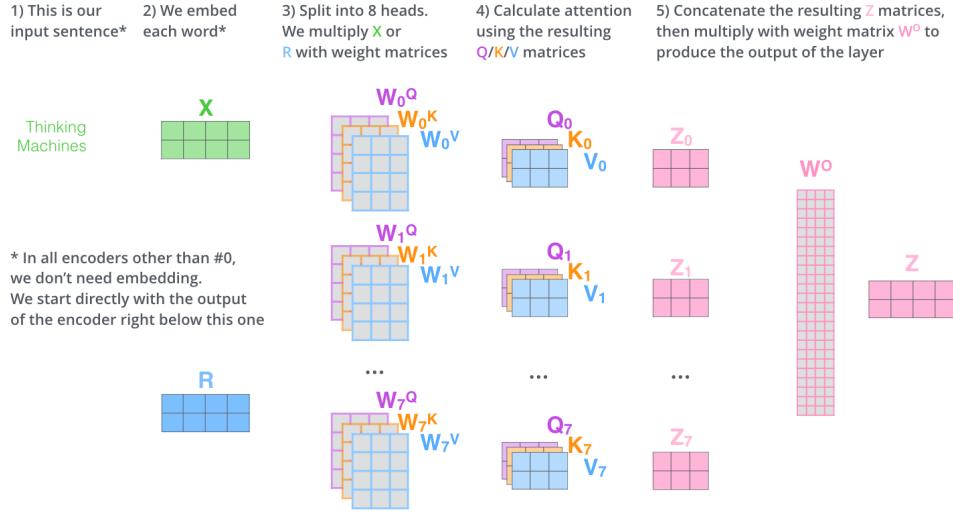
What are we actually doing when we do self-attention? In source-attention the semantics seemed clear; we're looking at corresponding words to be translated. But in self attention that's not the case. We are probably combining some semantic and syntactic coordination.

But there are different aspects of information we might want to attend. It seems odd to distill them down into a single (Q, K, V) triple. And since we noticed the similarity to CNNs we can use a technique used in CNNs: multiple filters! Indeed, we actually do attention in one place many⁵ times with a different learned (Q, K, V) set for each time; each attention that is learned is called a 'head'. Rather than use e.g. max-pooling or mean-pooling as is often done in CNN, Transformer instead does a linear projection of the heads (Alammar):



Here is attention all together (Alammar):

⁵eight



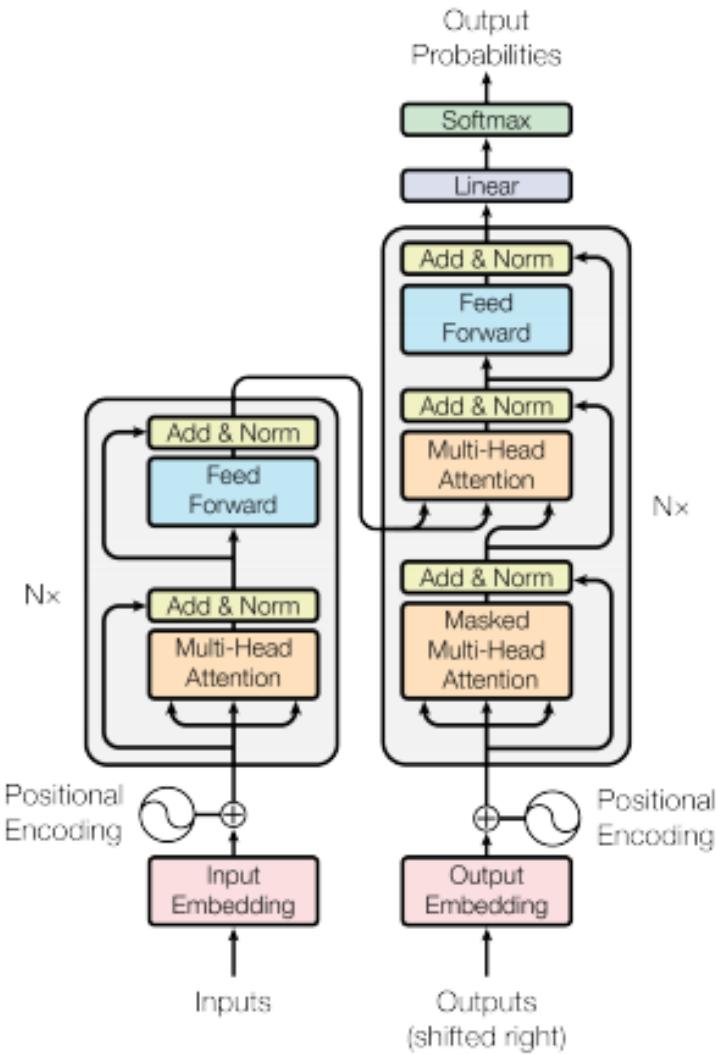
10.1.4 Residual Connections and Layer Norm

In the story we've told so far, data enters a layer, is combined with information from all the other words in the sentence (so far, for decoder) with self attention, if the decoder, is then combined with all the words in the source, then is projected through a feed-forward layer. So if we call the input to a layer x and the self-attention, source-attention, and feed-forward sublayers functions $self$, $source$, and ff , the output on the encoder is $ff(self(x))$ and on the decoder is $ff(source(self(x)))$. This seems like a good opportunity for the information at that position to get lost; self-attention could decide not to attend to the self! A well-known technique called *residual connections* is used; in each case we simply add the input back again after each sublayer. This is only done per-sublayer.

We introduce some sub-results: on the encoder we calculate $x' = self(x) + x$. Then the output is $ff(x') + x'$. Similarly on the decoder we calculate x' as before, then $x'' = source(x') + x'$ and the output is $ff(x'') + x''$.

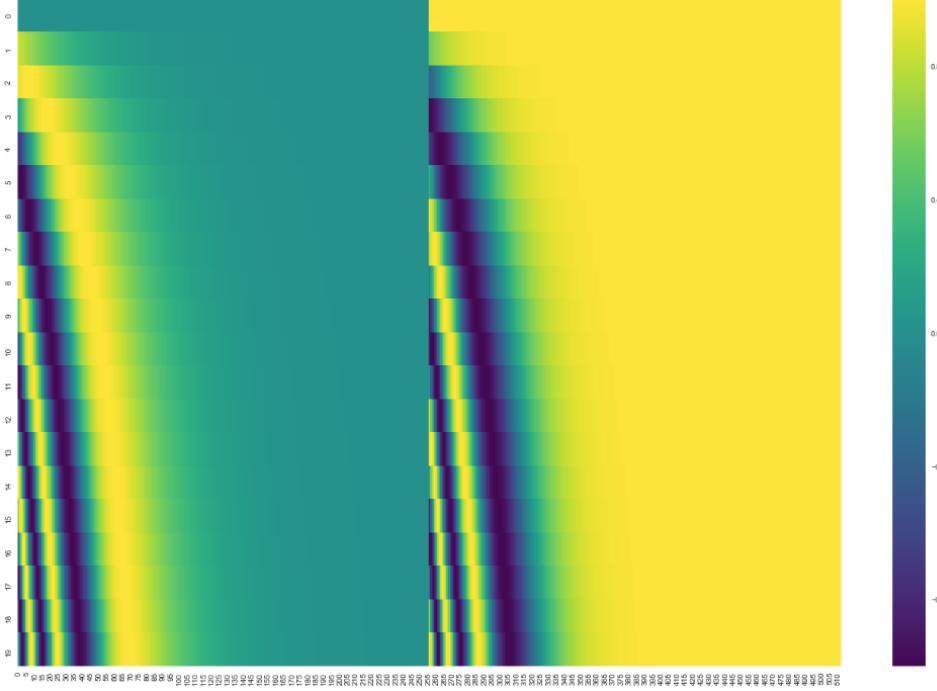
OK but we don't actually even use the original x or the other intermediates without modification either! Instead we use a technique called 'layer norm' [2] which essentially modifies each item by subtracting the mean and divides by the standard deviation over the vector.⁶ So in fact $x' = self(norm(x)) + x$, and for the decoder, $x'' = source(norm(x')) + x'$; For the encoder, the output is $ff(norm(x')) + x'$, and for the decoder it is $ff(norm(x'')) + x''$. This is not well-described in the original paper but is what has been uncovered (by TG and others). We now know almost everything in this handy diagram from the original paper:

⁶it's a little more complicated than that but this is already rather in the weeds.



10.1.5 Positional Embeddings

We haven't put any explicit notion of the ordering of the words in yet. So a specific sinusoidal function is added element-wise to each embedding. Specifically, for the n th word (0-based), in the even positions $2i$ of the embedding, for 1 to 512, $\sin(\frac{n}{10000^{2i/1024}})$ is added and in the odd positions $2i + 1$, $\cos(\frac{n}{10000^{2i/1024}})$ is added. The paper says that the position embedding for any $j + k$ can be represented as a linear function of the position embedding for j , so the other elements in the Transformer can take advantage of this, if they need to, and then there is theoretically no limit to the number of tokens that can be read. However in practice there are other ways to do this, and having an upper bound on length is not that big a deal. Here is what the position embedding look like:



10.1.6 Shared embeddings and BPE

One last part of the model: the word embeddings! Or should I say, the word *piece* embeddings. Before I explain that, notice there are three places where something like embeddings are used:

1. Source words are converted into embeddings
2. Target words are converted into embeddings
3. the matrix that makes logits looks an awful lot like an embedding (one dense vector for each vocabulary item).

This is three very large tables that all have the same dimension (though one is transpose); would be nice to have just one table. So we can do that – simply use the same parameters for all three cases. Now, you might think this is silly; the source and target vocabulary are quite different from each other! And that can be true, though in practice there is quite a bit of meaningful overlap, e.g. names, numbers, times.

Plus, if we break up the words we might have even more overlap! If both languages are in latin characters, we can just use characters instead of words and then there will be lots of overlap! But in Transformer instead, something in-between characters and words is used: byte-pair encodings (BPE): [28]. This is a kind of unsupervised word segmentation algorithm that works as follows:

```
def bpe(merges, vocab):
    for i in range(merges):
        # count all adjacent bytes, e.g.
```

```

# "four" = "f o u r" = "f/o, o/u, u/r"
# multiply by count of word in corpus
pairs = get_stats(vocab)
# find the most frequent pair. let's say it's "o/u"
best = argmax(pairs)
# now consider "ou" to be merged. So next time
# "four" = "f ou r" and instead you count "f/ou, ou/r"
vocab = merge_Vocab(vocab, best)
return vocab

```

You run it as long as you want. Instead of `merges` you can consider the size of the vocabulary you want. For latin languages the minimum is around 60 (e.g. most of ASCII) but this means the vocabulary blow up of before is no longer relevant. Furthermore, you can now more easily understand why it's useful to combine the embedding tables.

What about different character sets? Well, most people don't seem to worry about that because of hegemony. But we (at ISI) do; we have romanization tools that convert all langauge data into a common space (latin letters...still pretty hegemonic).

10.1.7 Model optimization tips

- Batches had 25k tokens (in practice similar length sentences are grouped together for maximum parallelism with minimal lost work).
- Adam optimizer (SGD with fancy learning rate) plus a fancy learning rate on top of that
- Dropout! Everywhere in the model, with probability 0.1, treat a parameter as if it was 0. don't contribute to the loss, don't update on backprop.

Chapter 11

Pre-Trained LMs

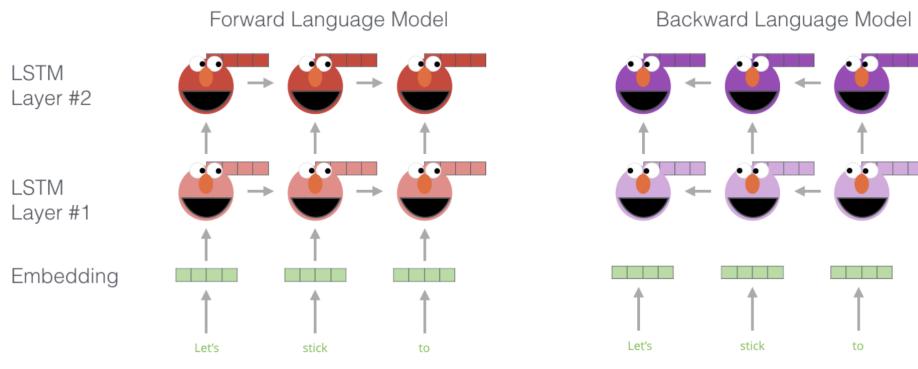
11.1 ELMo

In 2016, despite the rise of NMT and biLSTM-CRF, the predominant use of ‘neural networks’ in NLP was to insert type-based word embeddings like GLOVE or GenSim (implementation of Word2Vec) into existing models. ELMo [22] (Embeddings from Language Models) from AI2 came out in 2018 and introduced what it pitched as better embeddings. It showed across-the-board improvement on a number of diverse NLP tasks and was, not surprisingly, the best paper at NAACL, given that everyone knew about it by the time the conference came around (it was first posted in October 2017). The claim was that this is a set of *contextualized* word embeddings. That is, instead of having one representation for *bank*, the word has a different representation depending on the context (i.e. sentence) it appears in.

How is this done? Well, first a contextual model of text is needed. That’s easy, we’ve already seen several. This predates (sort of) Transformer, so ELMo used the predominant method at the time, bidirectional LSTMs.

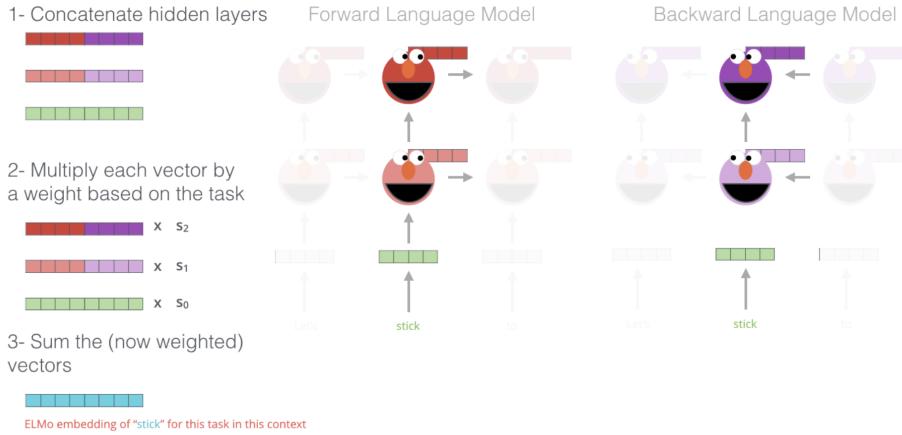
LSTMs are trained on plain text for the language modeling task, i.e. predict the next word (for the forward LSTM) or the previous word (for the backward LSTM). Here’s an illustration from The Illustrated BERT: ¹

Embedding of “stick” in “Let’s stick to” - Step #1



¹<http://jalammar.github.io/illustrated-bert/>

Embedding of “stick” in “Let’s stick to” - Step #2



This is trained on the Billion Word Benchmark [6] which is 1B English words from WMT 2011. That probably took a while but AI2 did it so you don't have to!²

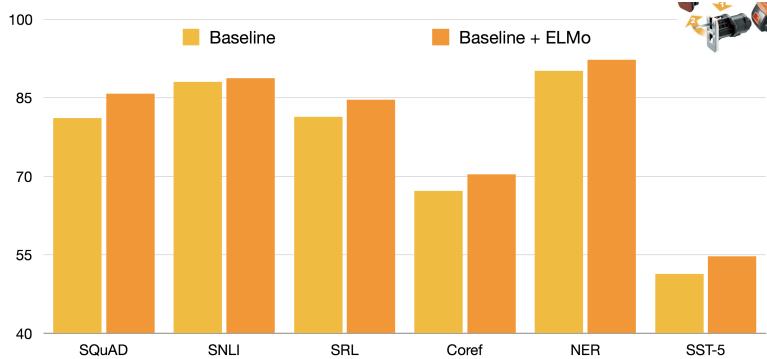
Now an embedding of a word in its context is obtained by running the context (i.e. the sentence) through the trained bi-LSTM and reading off the hidden state in both directions. Or, as it turns out, you can take some linear interpolation of hidden states at each layer; specifically how to linearly interpolate can be chosen by fine tuning interpolation parameters. However the core embeddings aren't fine tuned; they're just produced and used.

What was really cool about ELMo is you could use these embeddings in place of embeddings in your previously built models for various tasks and you pretty much got a gain. The most impressive results presented with the ELMo paper were across-the board lifts in the GLUE [31] tests by taking SOTA models and substituting in ELMo embeddings:

Task	Previous SOTA		Our baseline	ELMo + Baseline	Increase (Absolute/Relative)
SQuAD	SAN	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al (2017)	88.6	88.0	88.7 +/- 0.17	0.7 / 5.8%
SRL	He et al (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al (2017)	91.93 +/- 0.19	90.15	92.22 +/- 0.10	2.06 / 21%
Sentiment (5-class)	McCann et al (2017)	53.7	51.4	54.7 +/- 0.5	3.3 / 6.8%

Here's a bar graph (sam bowman slides)

²Note: how are the words initially embedded? The paper is pretty murky about this! Best I can figure they are read in as a character CNN (but I won't get into the details about this; somewhat also murky details are in [12] – this is what happens when you don't use peer review!!



I should probably mention what these tasks are:

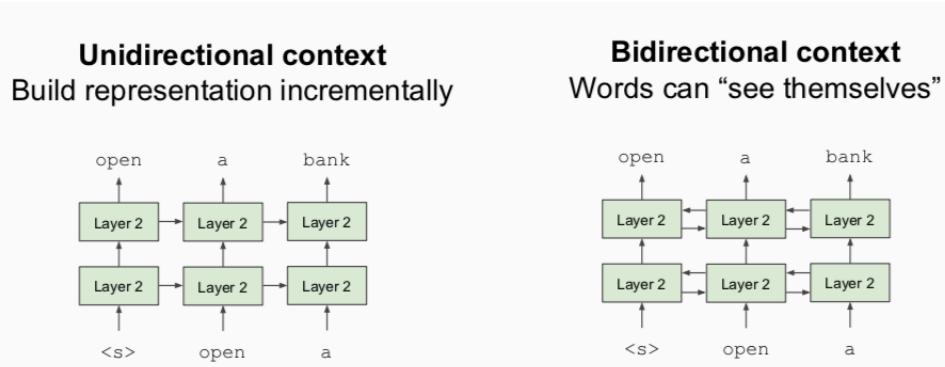
- SQuAD: question answering, extractive. Find the span.
- SNLI: natural language inference, aka ‘entailment’: given a pair of sentences (A, B), does B entail A, contradict A, or is it neutral to A? If A=‘Three men are standing in a field’ and B=‘People are standing’, B entails A. If B=‘People are sleeping’, contradiction. If B=‘The field is covered in snow’, neutral. Classify correctly.
- SRL: determine the semantic roles of text spans as they relate to verbs (e.g in ‘Mary sold the book to John’, Mary=agent, John=recipient, sold=predicate). Classification.
- Coref: Determine which mentions are of the same entity
- NER: find the spans and label with entity type
- Sentiment: classify sentence sentiment in a 5-way label

11.2 OpenAI GPT

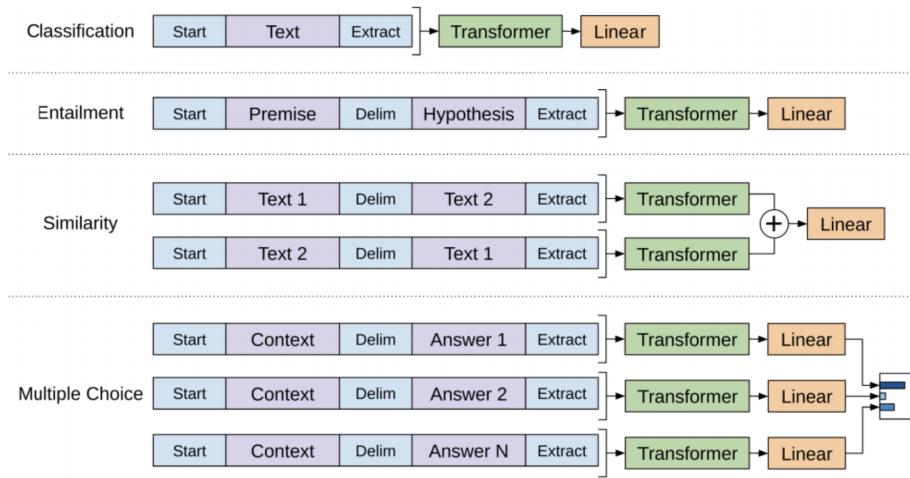
Not to be out done, in June 2018, OpenAI improved upon ELMo in a paper that IMO didn’t get too much attention [24], maybe because it wasn’t even put on ArXiv AFAICT, let alone submitted for publication. It had the following differences from ELMo:

- Transformer architecture instead of biLSTM. Along with that, using BPE.
- Designed directly for task prediction, with no other architecture, and carried with it a notion of *fine-tuning*; a task (e.g. multiple choice question answering) is turned into input sequences (e.g. question, separator token, answer choice). The topmost hidden unit after reading the last word is connected to a feed-forward classifier. Cross entropy on the classifier back-propagated.
- Trained on different data (Books corpus = 800M words)

GPT is essentially a Transformer *decoder* without source attention to an encoder. In other words, it uses masked self-attention that only looks to its left. If it didn’t, the topology of Transformer means it could ‘cheat’:



Here's an illustration of how task prediction works. You structure your input data as a series of sentences and then put a feed forward/linear layer on the end to map to classification.



I didn't actually hear GPT until reading the BERT paper...maybe BERT had better marketing.

11.3 BERT (images from Jacob Devlin slides)

ELMo had a few months of glory (and everyone(?) ignored GPT) until October 2018 when Google struck back with BERT (Bidirectional Encoder Representations from Transformer) [8], clearly riffing on the muppet theme.³ Like GPT, BERT used Transformer and subwords (though it used google's slightly different WordPiece [32]). BERT also used the fine tuning paradigm. But there were more important differences:

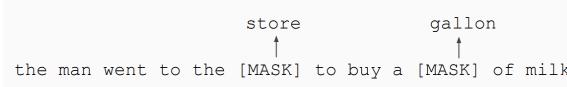
- New objectives: Bidirectional prediction using word masking and next sentence prediction
- More structured two-sentence representation, class token for predictions included during training (first word of every input is the otherwise unused [CLS]).

³Yes, there were more muppet themed papers: GROVER (Generating aRticles by Only Viewing mEtadata Records.) [33], ERNIE (Enhanced language RepresentatioN with Informative Entities) [36] (I think there were two ERNIES actually). There was something branded ‘big bird’ but it wasn’t part of the paper name. The trend seems to have eased, thankfully.

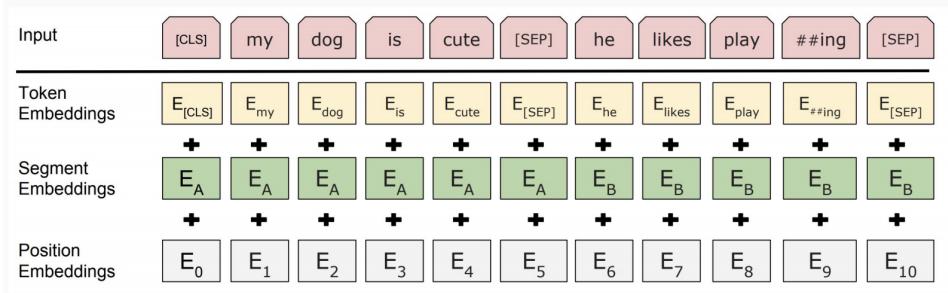
- Pretraining+Fine Tuning recipe
- Trained on a lot more data (Wikipedia = 2.5B words + Books corpus = 800M words)
- There's a large version of BERT with tons of parameters: for L=layers, H=hidden units, A=attention heads, BERT-BASE = (L=12, H=768, A=12, Total Parameters=110M) = same size as GPT; BERT-LARGE = (L=24, H=1024, A=16, Total Parameters=340M)

In ablation studies, the BERT authors claim the key is in the pretraining tasks: GPT and ELMo just pretrained on the language model objective (predict next word).

To pretrain, BERT masks out 15% of the words from its training data and then tries to predict them (15 seemed to be the magic number):



BERT also structures its input in the following way:

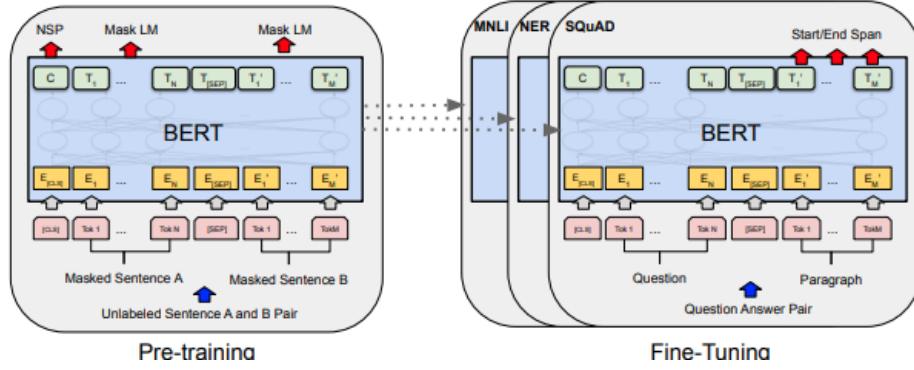


An encoding value is learned (same value on each position) for ‘sentence 1’ vs ‘sentence 2’ and added to each embedding. This is how data is then set up (see above). The [CLS] token is used instead of the last word token used in GPT.

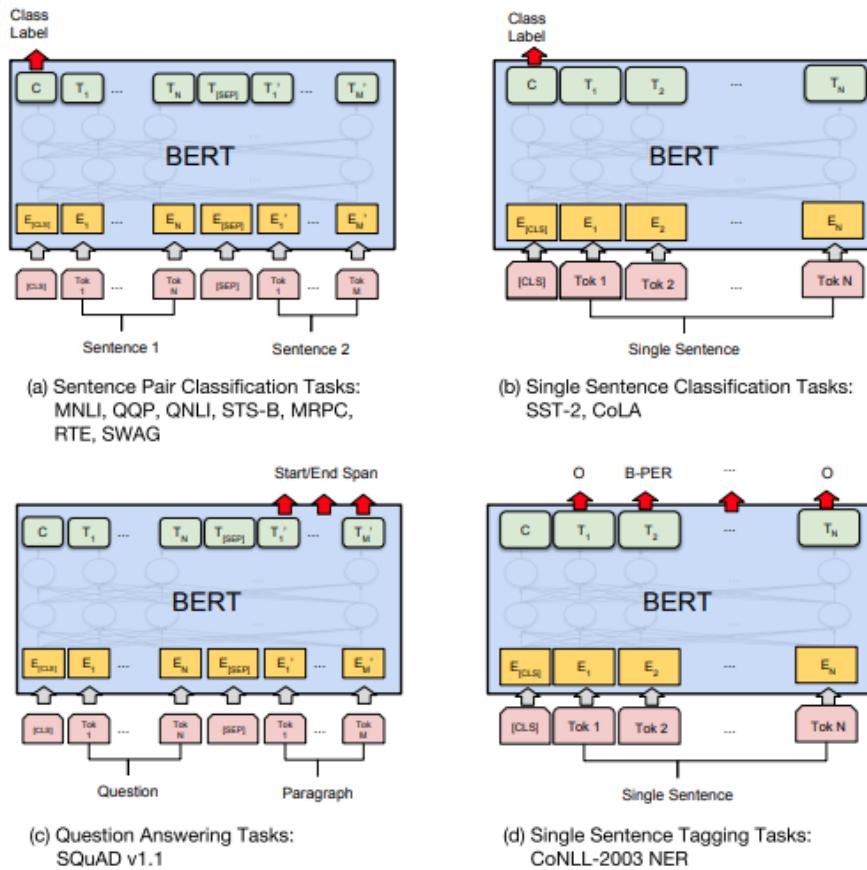
Two pretraining losses are calculated. For each MASK token, the top level hidden unit corresponding to each MASK predicts a word from the vocabulary (well, loss for probability of the correct word is calculated). Only sometimes (10%) a random word is used instead of [MASK] and sometimes (10%) the right word is used, but the 15% of words we need to predict in this pretraining are specified in the training corpus. Note that now self-attention can span the entire sentence.

Additionally, a next sentence prediction task is used: Either sentence 2 is the next sentence or it isn’t, and this is learned by feeding the top hidden unit for [CLS] into a binary classifier.

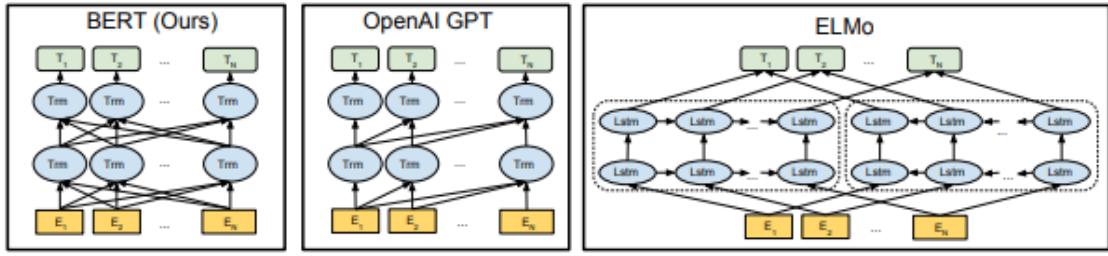
Apart from pre-training, BERT uses per-task fine-tuning. Here’s a diagram from the BERT paper comparing the two:



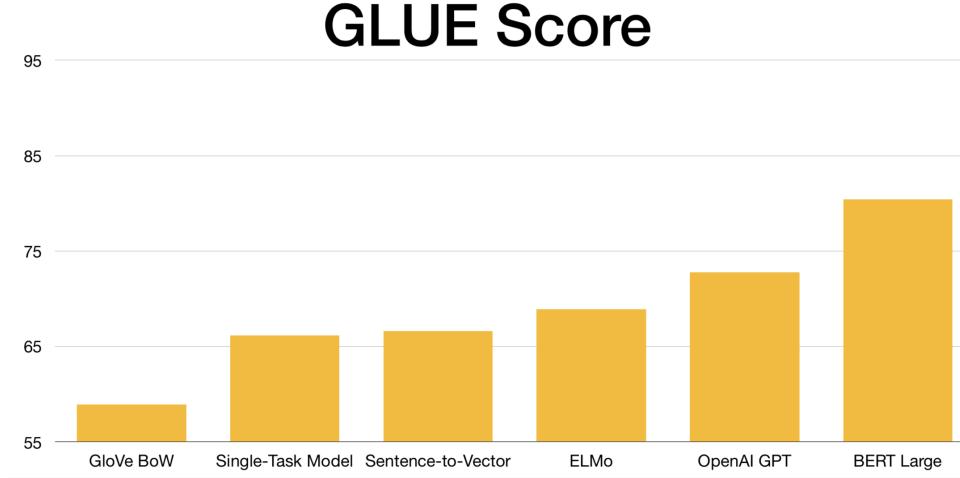
Fine tuning is the same idea as before, though BERT can be used both in classification and tagging paradigms (so could the other models, presumably). Here are the setups (from BERT paper):



Here's an overview comparing the topologies of ELMo, GPT, and BERT (from BERT paper):



Results were significant:



System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
BERT _{BASE}	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{LARGE}	86.7/85.9	72.1	91.1	94.9	60.5	86.5	89.3	70.1	81.9

The tasks:

- MNLI: like NLI but done over many genres, supposed to be less biased (we'll get into that)
- QQP: Quora question pairs: given two questions, are they asking the same thing?
- QNLI: SQuAD converted into a binary NLI task (does this sentence answer the question?)
- SST-2: Binary sentiment analysis
- CoLA: Given an english sentence, is it ‘acceptable’ to native ears (‘Bill’s book has a red cover.’) or not (‘The Bill’s book has a red cover.’)
- STS-B: Sentence pairs annotated with score from 1 to 5 on semantic similarity
- MRPC: Are two sentences semantically equivalent?

- RTE: Like MNLI but much less training data

An additional GLUE task, WNLI, is the Winograd challenge (Resolve ‘it’ in ‘The trophy didn’t fit in the suitcase because it was too big/small.’) At BERT publication no model, including BERT, outperformed majority baseline (65.1). (This has since changed.)

A quick followup from Facebook, RoBERTa (Robustly Optimized BERT pretraining Approach) [18]:

- Even more data. Everything in BERT (Book Corpus and Wikipedia = 16GB uncompressed) plus Common Crawl news (76 GB after filtering) plus web text data linked to from reddit with 3+ upvotes (38 GB) plus a subset of common crawl filtered to look like winograd stories (31 GB).
- Unlike BERT, masking was done multiple times on sentences.
- Next Sentence Prediction as described in the BERT paper (but possibly not in the implementation) seems to hurt, so it was removed. Just masking is used. (So what happens to the CLS token training? Presumably only fine-tuning is used to make it meaningful but this is somewhat unclear to me.)
- Using the same training settings as BERT, and the same data, RoBERTa was better. When adding more data and training even longer it was even better.

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
<i>Single-task single models on dev</i>										
BERT _{LARGE}	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
XLNet _{LARGE}	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	90.2/90.2	94.7	92.2	86.6	96.4	90.9	68.0	92.4	91.3	-
<i>Ensembles on test (from leaderboard as of July 25, 2019)</i>										
ALICE	88.2/87.9	95.7	90.7	83.5	95.2	92.6	68.6	91.1	80.8	86.3
MT-DNN	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2/89.8	98.6	90.3	86.3	96.8	93.0	67.8	91.6	90.4	88.4
RoBERTa	90.8/90.2	98.9	90.2	88.2	96.7	92.3	67.8	92.2	89.0	88.5

There have since been many more:

- DistilBERT [27]: Almost as good as BERT but a lot faster and smaller
- ALBERT [15]: A Lite BERT. same idea.
- BART [17] BERT but a sequence-to-sequence model, useful for generation and classification
- T5 [25] really big Transformer trained on Common Crawl filtered for English, then fine-tuned on a lot of tasks all at once
- Multi-language versions of these
- Domain-specific versions of these

11.4 RoBERTa

In August 2019 Facebook struck back with RoBERTa (Robustly Optimized BERT pretraining Approach) [18].

Key differences:

- Even more data. Everything in BERT (Book Corpus and Wikipedia = 16GB uncompresssed) plus Common Crawl news (76 GB after filtering) plus web text data linked to from reddit with 3+ upvotes (38 GB) plus a subset of common crawl filtered to look like winograd stories (31 GB).
- Unlike BERT, masking was done multiple times on sentences.
- Next Sentence Prediction as described in the BERT paper (but possibly not in the implementation) seems to hurt, so it was removed. Just masking is used. (So what happens to the CLS token training? Presumably only fine-tuning is used to make it meaningful but this is somewhat unclear to me.)
- Using the same training settings as BERT, and the same data, RoBERTa was better. When adding more data and training even longer it was even better.

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
<i>Single-task single models on dev</i>										
BERT _{LARGE}	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
XLNet _{LARGE}	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	90.2/90.2	94.7	92.2	86.6	96.4	90.9	68.0	92.4	91.3	-
<i>Ensembles on test (from leaderboard as of July 25, 2019)</i>										
ALICE	88.2/87.9	95.7	90.7	83.5	95.2	92.6	68.6	91.1	80.8	86.3
MT-DNN	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2/89.8	98.6	90.3	86.3	96.8	93.0	67.8	91.6	90.4	88.4
RoBERTa	90.8/90.2	98.9	90.2	88.2	96.7	92.3	67.8	92.2	89.0	88.5

11.5 BART?

It never stops! On October 29 2019 (a week before I wrote these notes) Facebook released BART [17]. From an early look at the paper it is more of a denoising auto encoder structure (i.e. translate noisy English into clean). It seems to be close to RoBERTa on the tasks discussed but maybe better on generation-based tasks.

11.6 HuggingFace

A major aid to experimentation is HuggingFace (<https://huggingface.co/>) which has come to prominence by making these models and others not discussed here available as pretrained PyTorch with common user interfaces. They are relatively easy to use and it's easy to compare different models and see which works best on your task. Check them out

at <https://github.com/huggingface>. (I have no relationship with this company, I'm just happy they've made my students' lives easier!)

11.7 Blade Runner NLP

In the movie Blade Runner, a hunter named Decker tried to find and terminate rogue robots that were nearly indistinguishable from humans (see also the reboot of Battlestar Galactica, the TV series The Americans, and numerous other Sci-Fi and non-sci-fi treatments of this idea). He administers a test called the 'Voight-Kampff Test' where he asks questions that AI, lacking full human sentiment and experience, could not answer 'properly' (and might have an unrealistic skin/heart/iris response to). We might say that the GLUE tasks and some others have become attempts to detect AI in models like the ones described above. The next lecture goes through a selection of slides from Sam Bowman who has been looking deeply at this problem. Here's an example Voight-Kampff:⁴

5 While walking along in desert sand, you suddenly look down and see a tortoise crawling toward you. You reach down and flip it over onto its back. The tortoise lies there, its belly baking in the hot sun, beating its legs, trying to turn itself over, but it cannot do so without your help. You are not helping. Why?

- What do you mean, I'm not helping?
- What is a tortoise?
- I don't know why I would flip the turtle over in the first place

⁴<https://www.allthetests.com/quiz30/quiz/1386100782/Voight-Kampff-test-questions>

Chapter 12

Evaluation & Annotation

Nearly all of NLP benefits from a dispassionate analysis of how well the models we build perform the analysis or generation of NL data we are trying to have them analyze/generate. Aside from avoiding inherent bias and blindness toward a system's quality (which can lead to trouble down the road), there are circumstances where optimizing on the metric can be a good way of improving performance. But understanding what correctness and incorrectness is and how to measure these values can be tricky. Here we try to cover some of the key methods of evaluation in NLP. Evaluating these models means having labeled data for them; we'll discuss strategies for collection too.

12.1 Quantitative Analysis

12.1.1 Development, Test, Blind

One way to build your system is to look at some labeled data, write code that tries to get the right labels on that data, see what labels it's getting incorrect, refine, repeat. At the end you will have a system that does well on the corpus you looked at. But this often means you will have neglected some phenomena present **outside** your training corpus. Worse, you may engineer things so well on training that previously correctly labeled items outside of your training corpus are now mislabeled.

This is called *overfitting* and it usually means you are taking advantage of some eccentricity in your training data that does not generally hold (e.g. all sentences with an even number of words are positive sentiment).

To avoid overfitting it's a good idea to divide your data as follows: most (80-90%) is used to build your model (train corpus), a sample (10-20%) is used to periodically evaluate but not to build the model (dev corpus), and another sample (5-10%) is not looked at and only used for evaluation, very very seldomly (test corpus). You can track overfitting: plot a graph of training vs. dev performance over time; if dev starts to go down while train goes up, that's overfitting. How much to evaluate on test? It's an art, honestly. The more you do, the more you will probably overfit, and then you'll need another test corpus to verify this.

If you don't have a lot of data, you can do what's called *cross-validation*. You divide up your data and evaluate. Then you slide the dividing lines to create another *fold* of the

data. Keep doing this and every chunk of data gets to be train, dev, and test. Then average the results. I think this has the tendency to lead to overfitting more quickly but it is not infrequently used.

The extreme version of cross-validation is n -fold where n is the number of items you have; this is called *leave-one-out* validation and allows the maximum amount of training data to be used (but I trust it the least).

A hybrid strategy, where cross-validation is used for train and dev, but the test set is held constant, is probably a good compromise.

The best scenario is when someone else holds on to a piece of the data for you and you only ever get to see it once, when you're totally done with your model. This corpus is called 'blind' and is usually only used in the context of shared tasks.

It's important to consider how you divide your data. Ideally data should be independent and identically distributed (IID). You might think random selection of labeled elements would be suitable. For a collection of sentiment-labeled movie reviews this is true. However, if your corpus is a set of documents, you don't want to have one sentence from a document in training and another in test. Words and phenomena tend to cluster in a document, since a document is about some topic, with topic-relevant words, and is generally written by one author, and will have a particular style. So best practice is to divide up along *document* boundaries randomly, but pay attention to the number of evaluated items that get added to each set.

Some notes on evaluation measures follow. The choice of evaluation measures is always subject to debate (look at MT metrics workshops) but here are some guidelines I like to use:

12.1.2 Accuracy

For strict classification, where each item receives a label from a fixed set of labels, and the distribution of labels is reasonably even (doesn't need to be all the way even, but shouldn't be 90% one class), simple accuracy = $\frac{\text{correct}}{\text{total}}$ is a perfectly good metric.

12.1.3 F-Measure

For cases where there is one 'background' label that predominates and a relatively few instances of a 'content' label (e.g. named entity recognition) F-measure, which combines precision and recall, is a better choice, and is calculated on all labels *except* the background label. Precision is $\frac{\text{correct}}{\text{hypothesis}}$, i.e., how much of what you predicted is correct. Recall is $\frac{\text{correct}}{\text{reference}}$, i.e. how much of what was correct did you predict. Using either one by itself can be misleading: why?

F_1 is a *harmonic mean* of precision and recall. Specifically it's $2 \cdot \frac{PR}{P+R}$. In general $F_\beta = (1+\beta^2) \frac{PR}{\beta^2 P + R}$; F_2 is weighted to favor recall, and $F_{0.5}$ is weighted to favor precision. In certain circumstances one may be preferred (e.g. precision is less important if there will be a downstream selection task, while recall is less important if the task is to mine information from a very large corpus and the amount of information mined is more important than assurances it has all been mined).

What is described above can be more specifically described as *micro-averaged* F1, i.e. each non-background labeling is considered in making the tallies of what is correct, then the averages are calculated. One can also consider *macro-averaged* F1, where F1 is calculated for each kind of label (e.g. in named entity recognition, PERSON, then LOCATION) where all other labels are considered to be background. These F1s are then averaged together.

Micro-F1 is the same as accuracy when there is no background class; Macro-F1 can be done in these situations too if there is class imbalance (i.e. one dominant class is mostly correct but you want to weight each class evenly instead of each item) Example:

		Gold				
		None	Person	Location	Company	Total
Hypothesis	None	N/A	0	5	10	15
	Person	0	200	10	0	210
	Location	0	5	40	0	45
	Company	5	0	0	10	15
	Total	5	205	55	20	

$$\text{Micro-averaged Precision: } \frac{200+40+10}{270} = 0.926$$

$$\text{Micro-averaged Recall: } \frac{200+40+10}{280} = .893$$

$$\text{Micro-averaged F1: } 2 \frac{.926 * .893}{.926 + .893} = 2 * .827 / 1.819 = .909$$

$$\text{Macro-averaged precision: } \frac{\frac{200}{210} + \frac{40}{45} + \frac{10}{15}}{3} = .836$$

$$\text{Macro-averaged recall: } \frac{\frac{200}{205} + \frac{40}{55} + \frac{10}{20}}{3} = .734$$

$$\text{Macro-averaged F1: } 2 \frac{.836 * .734}{.685 + .854} = 2 * .614 / 1.539 = .782$$

12.1.4 Granularity

It's important to consider what item is ultimately judged correct or incorrect. Even if each word or sentence is labeled, it may make more sense for a sequence of adjacent labels to be examined to consider whether an item is correct or not.

12.1.5 Rank-based evaluation

Sometimes you get to return more than one label, or you get to label your items in preferential order (e.g. information retrieval). Sometimes your results are actually pretty bad but you want to convey that your algorithm is better than random (e.g. unsupervised bilingual lexicon induction). There are a few strategies for evaluating this kind of data:

Precision/Recall@N: Consider up to N ranked items. Careful – the way this metric is implemented can vary! In e.g. information retrieval, P@N means you consider N items per query and calculate precision over the $N*Q$ total items retrieved. However in e.g. bilingual lexicon induction, P@N means if *any* of the N items retrieved is correct you consider the entire item is correct, i.e. precision is calculated over Q total items retrieved. If P@N monotonically increases with N , it's the latter.

Precision@R: Especially in retrieval like cases, you can control the tradeoff between more precision and more recall by adjusting your threshold of returning an item. This would numerically show the precision at a fixed recall. This can also be plotted to determine ideal

operating points. A calculation of the *area under the curve (AUC)* is a good single number that summarizes this tradeoff (higher is better).

12.1.6 Edit distance/word error rate

For structured output, especially text that is generated, the idea of ‘how much work does it take to fix this’ is very relevant, especially if the task will be, say, a first pass before human correction (this is very common in, say, the translation service industry). Given the number of substitutions S , deletions D , and insertions I made over a text of length n , the error rate (WER or TER for task T) is:

$$WER = \frac{S + D + I}{n}$$

Variants assign different cost to different operations or operations involving different components. For example, inserting/removing determiners might only cost 0.4 of an edit, while doing so for content words could cost 2.3. These values are set experimentally.

12.1.7 Intrinsic Vs. Extrinsic Analysis

Intrinsic: evaluate the task on its own merits. E.g. parse F1, POS tag accuracy. Evaluation ‘close’ to the task. Advantage: directly evaluates the model you’re building. Disadvantages: does it matter?

Extrinsic: evaluate the task as it plugs into some other task (e.g. parsing in the service of summarization). Need to hold the other technology constant. There are different levels of this (narrow = pos tagging for parsing; wide = machine translation for surgery outcome). This also brings up the question of **component** vs. **end-to-end** evaluation. Should you use a sequence of noisy components (more realistic, exposes problems of interaction, can be tough to tell what actually causes the problem) or use ‘gold’ data at every point in the pipeline before your tool (better for narrow debugging, doesn’t give a realistic picture)? Both have their value. Don’t let something out before testing end to end, though.

12.1.8 Human judgments

The gold standard! Just like with machines, you have to tell humans how to evaluate. And if the way you tell them to evaluate is too ‘weird’ you may not get results you like.

The major advantage is you can ask for judgement calls that are specifically something machines can’t do. For example:

- Given one reference translation, ask them to edit a machine translation until it *means the same thing* as the reference.
- Evaluate the sentiment of a text on a n -point scale
- Is a response sarcasm?
- ...

There are a few drawbacks:

- Humans are slow compared to machines.
- Humans are expensive compared to machines.
- Humans are inconsistent.

The last one is maybe trickiest. The same human may give two different annotations. Or two humans may be internally consistent but disagree with each other. There are ways to improve agreement:

- Have several annotate and take the most frequent label (hopefully small label set)
- Have several annotate, then talk together and resolve differences
- Have several annotate, then have a super annotator resolve differences.

Ultimately, though, you want to track *inter-annotator agreement* (IAA):

Basic idea: ask n humans to annotate the same data. Check for their overlap. There are several metrics, most based on this basic equation: $\frac{P(a) - P(e)}{1 - P(e)}$ for agreement a and expected agreement e . 1 for perfect agreement. 0 for chance agreement. Can be negative if agreement is worse than chance.

- Cohen's κ (only good for 2 humans): See below. What is a good value? Wide disagreement. If you have to choose, at least 0.8 (but some might say 0.4).
- Scott's π – same as Cohen's κ but take squared arithmetic means to determine $P(e)$. Less informative than Cohen's since it assumes equal distribution of responses.
- Fleiss's Kappa (generalizes to n humans)...maybe 0.5 is ok? Generalization of Scott's π .
- Krippendorff's α – more general. Also allows for missing data, partial agreement. Quite complicated and computationally intensive to calculate.

Demo:

		B	B	B	total
		pos	neut	neg	
A	pos	34	8	0	42
A	neut	51	38	26	115
A	neg	0	21	72	93
	total	85	67	98	250

A used 'pos' 42 times = $42/250 = .168$. B used it 85 times = $85/250 = .340$. So the expected probability of agreement for pos is $.168 \times .340 = .05712$. Add up the probability of chance agreements to get total $P(e) = .05712 + .12328 + .145824 = .326224$. $P(a) = \frac{34+38+72}{250} = .576$. Then $\kappa = \frac{.576 - .326224}{1 - .326224} = .3707$.

For Scott's π but instead we calculate $P(e)$ as $\frac{42+85}{500}^2 + \frac{115+67}{500}^2 + \frac{93+98}{500}^2 = .342936$ so
 $\kappa = \frac{.576 - .342936}{1 - .342936} = .3547$

Note: IAA not really helpful for, e.g., translation. There you just want to collect many different responses and use them all to evaluate.

Which to use? Often several are used together. Cohen's over Scott's, but Fleiss' if needed, α if extra mechanism needed. Statisticians might know more.

12.1.9 Statistical Significance

Ok, you have some results, automatic or human. But can you rely on them? Questions you can ask yourself:

- Are the judgements actually measuring something real?
- Are they something that we care about?
- Is it from the domain/genre that we care about?
- Is it from the right distribution?
- Are there enough examples that we can trust it?

The last question is something we can answer. See also section 4.4.3 of Eisenstein and the Berg-Kirkpatrick reading this narrative is taken from:

Let's say we have two classifiers, A and B. A is better than B on some test set x by $\delta(x)$. Null hypothesis H_0 : A is not actually better than B. If true, how likely is it that A would be better than B on some new set x' by at least $\delta(x)$? If $P(\delta(x') > \delta(x) | H_0) < 0.05$ ¹ then we can say with 95% confidence that H_0 is rejected and indeed A is better than B. 0.05 is called the p -value.

There are a variety of methods for establishing p-value, but one easy way that works for lots of metrics and situations where we don't have limitless test data is the following bootstrap approach (Berg-Kirkpatrick et al. 2012):

1. Draw b bootstrap samples y of size n from x with replacement.
2. Let s be 0
3. for each y , if $\delta(y) > 2\delta(x)$, increment s
4. $p \approx s/b$

What's going on here? Since the samples are all drawn from the test set we in fact want to show how often A is better than expected, and it's expected to beat B by $\delta(x)$ since we already know it does. In the original presentation of this work, $b = 10^6$ which showed stable behavior of p calculation. This was done over a wide variety of task types.

¹Actually a random variable \mathbf{X} is used, not x'

12.2 Annotated Task Corpora

If you want to know how well you're doing on a task you should compare yourself to examples of the task done correctly. A catch-all term for this is an ‘annotation’ (also a ‘labeling’) of data. Here are some examples:

task	input	label
POS tagging	The boy ran home	DT NN VBD NN
constituency parsing	The boy ran home	(S (NP DT NN) (VP VBD NN))
dependency parsing	The boy ran home	2-det 3-nsubj 0-root 3-advmmod
sentiment	The boy ran home	neutral
translation into french	The boy ran home	Le garçon a couru à la maison

Some other kinds of annotation (the scope is limitless):

- phonetic: how was a word spoken, intoned, where were pauses taken, what words were stressed
- semantic: mark words as they’re used with senses, draw semantic relatedness graphs
- pragmatics/discourse: what role does each turn play (e.g. acknowledgement, request for feedback, acceptance, marker for new phase). Is a statement a thesis, antithesis, elaboration, rebuttal, justification, etc? what are the discourse units? Resolve anaphora – what do pronouns refer to? markers for attribution (something someone else claimed happened), For more look at rhetorical structure theory (RST)

12.3 How to Gather Annotation

The lack of data has stymied many a project but I encourage you to think of it as an opportunity, not a roadblock! Advice: “If you want to get a lot of citations, publish a corpus.” (Philipp Koehn, prof. JHU. Very famous. Lots of released corpora.)

12.3.1 Found

Sometimes there is natively an annotation already in existence, though possibly not in quite the right format. A simple case is that text reviews of movies often come with star ratings, which can be turned into positive/negative sentiment. A more esoteric kind of found annotation would be, say, using text spans in wikipedia that also contain page links that have info boxes used for celebrities to determine the presence of a person mention in NE tagging. Such methods can be quite powerful but are also quite noisy; they are often referred to as ‘silver standard’ for this reason.

Beware of using a found annotation that itself was programmatically generated or that you programmatically generate from the data, this is circular reasoning and will lead you to either create a trivial data set (e.g. list of descriptions of Japanese comic characters with

annotation of whether they were originally anime or manga; labels generated by finding which word comes first in the description) or one that is impossible to label (sentiment labeled data; labels are chosen by running sentiment analysis). It sounds absurd but is easier to accidentally create than you might think!

12.3.2 DIY

You can (and should) annotate some data yourself before trying to get other people to do it. This will help you develop your guidelines and give you a sense of how difficult the task is. However, it will still be an overestimate, because you may try to write down annotation guidelines (see below) but there will be hidden assumptions that won't come out until you try to have someone else do the same annotation you're doing

12.3.3 Phone a Friend

Just doing your own annotation is unwise; you could do consistent annotation but nobody would be able to follow on since you won't need to write a comprehensive standard. You also will inherently be observing any test data you produce, so your systemns will be likely to overfit. Asking someone to try to do the annotation (advisor, colleague in your lab) is a good first test of your approach. It also gives you a way to judge IAA. If they are good enough and the annotation is simple enough you can make a sufficiently sized corpus without too much effort, you can have your friend's annotations serve as a test set, and maybe they can be a coauthor of your paper.

12.3.4 Hire

Once you want to get serious about annotation you'll want to parallelize and increase your throughput rate and diversity of annotator. Direct hiring (we can sometimes bring in MS or undergrad student workers) has the main advantage that you can have fairly tight control over your workers' outputs, you can have them use custom tools quite easily, and you can get and give a lot of feedback. If your annotation has special skill requirements (e.g. knowledge of a particular language) this may be the best way to go. The downside is it can be tough to find dedicated annotators, since most annotation is rather tedious. It can also be expensive; you'll be competing with other employers and your team is likely to be able to choose where to work.

12.3.5 Crowd

Amazon Mechanical Turk and Figure 8 (formerly CrowdFlower) have made the job of hiring pools of annotators much easier, but there are a number of caveats when working with MTurk. First, it is not in practice as cheap as you might imagine. Ethically you should pay a living wage; we use \$15/hr as a minimum. You will get questions about this when you publish and you may get pushback. You will also get lower quality work or no takers if your rate is too low. In practice you can't pay hourly, you have to pay by the piece. This means your annotators will have much less appetite for reading very long annotation

guidelines. It also means you need to calibrate your pay by getting test annotation and estimating completion time.

Another major hassle is that there will be attempts to exploit you by not doing meaningful work. Although you are allowed to not pay for work that is not done, rejecting work already done for low quality will earn you a bad reputation, which will lower the quality and number of workers. It is important to carefully vet workers by having them randomly annotate items you already have the answers to and then validating their responses against these. You can also have workers do an entire set you know the answer to, and if they do well, you can give them specific qualification to do more work. For bots/non-workers, best bet is to simply not hire them again.

If you are doing research in a lab, annotation via crowd work may be considered human subjects research and require from one or more internal review boards (IRBs). This is more of an issue for your PI than you but you should discuss it before proceeding.

Finally, your interaction with workers is more limited. The interface options are smaller and conveying subtle differences in annotation standards can be tricky.

Caveats aside, using crowd workers is actually quite useful and sometimes the only way to get annotation work done. It's a good idea to try doing some crowd work to get a feel for it!

12.3.6 Inter-Annotator Agreement

While annotating you'll want to get IAA, in exactly the same way as you'd get IAA for scores per above.

12.4 Annotation Guidelines

It is generally a good idea to write down your intended rules for how to annotate, in as plain a language as possible. Note that this need not and should not be essentially a computer program (otherwise it wouldn't be an interesting NLP task) but can often be quite detailed. The part of speech tagging guidelines for the Penn treebank, for example, are 37 pages long! Guidelines are only effective if they're followed, of course, so you have to judge how much time your annotators will put in learning how to annotate. If you've hired them, with an hourly wage, longer manuals will probably be okay. If they're doing piece-work, e.g. in mechanical turk, they aren't going to spend time reading manuals and not getting paid.

Here are some classic examples. From the (37 page) PTB POS tag guidelines:

Adjective, superlative–JJS

Adjectives with the superlative ending *-est* (as well as *worst*) are tagged as JJS. *Most* and *least* when used as adjectives, as in *the most or the least mail*, are also tagged as JJS. *Most* and *least* can also be tagged as JJS when they occur by themselves; see the entries for these words in Section 5. Adjectives with a superlative meaning but without the superlative ending *-est*, like *first*, *last* or *unparalleled*, should simply be tagged as JJ.

Typically, part way through the annotation, tricky cases will be uncovered, then resolved, and added to the annotation standards, like was presumably done here:

Words that refer to languages or nations, like English or French, can be either adjectives (JJ) or proper nouns (NNP, NNPS).

EXAMPLES: English/JJ cuisine tends to be uninspired.
The English/NNPS tend to be uninspired cooks.

In prenominal position, such words are almost always adjectives (JJ). Do not be led to tag such words as proper nouns just because they occur in idiomatic collocations.

EXAMPLES: Chinese/JJ cabbage; Chinese/JJ cooking
Welsh/JJ rarebit; Welsh/JJ poetry

However, note:

EXAMPLE: an English/NP sentence
(cf. a sentence of English/NP)

Chapter 13

HMMs: POS Tags Case Study

13.0.1 What are Part-of-Speech (POS) Tags?

Syntactic labels of words: This/DET is/VB a/DET simple/ADJ sentence/NOUN. These abstract away from the core word meanings. Sequences (i.e. legal ordering) are mostly (though not entirely) dependent only on the labels, not by the words themselves.

- Open class words ("content words")
 - nouns, verbs, adjectives, adverbs
 - mostly content-bearing. refer to objects, actions, features in the world
 - open class = there is no limit to what they are or can describe so new ones are added all the time (email, website, defenestrate)
- Closed class words ("function words")
 - pronouns, determiners, prepositions, connectives
 - there are a limited number of these
 - mostly functional: to tie the concepts of a sentence together

How many? It depends. Some kind of annotation standard is needed to decide, e.g., should proper and common nouns be separated? Should singular or plural nouns? Present/past/main/aux verbs?

Penn treebank: fairly detailed set of tags (45 of them) used frequently along with parsing (particularly constituent parsing). Some weird quibbles: why does 'to' get its own tag?

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	<i>and, but, or</i>	SYM	symbol	+%, &
CD	cardinal number	<i>one, two</i>	TO	"to"	<i>to</i>
DT	determiner	<i>a, the</i>	UH	interjection	<i>ah, oops</i>
EX	existential 'there'	<i>there</i>	VB	verb base form	<i>eat</i>
FW	foreign word	<i>mea culpa</i>	VBD	verb past tense	<i>ate</i>
IN	preposition/sub-conj	<i>of, in, by</i>	VBG	verb gerund	<i>eating</i>
JJ	adjective	<i>yellow</i>	VBN	verb past participle	<i>eaten</i>
JJR	adj., comparative	<i>bigger</i>	VBP	verb non-3sg pres	<i>eat</i>
JJS	adj., superlative	<i>wildest</i>	VBZ	verb 3sg pres	<i>eats</i>
LS	list item marker	<i>1, 2, One</i>	WDT	wh-determiner	<i>which, that</i>
MD	modal	<i>can, should</i>	WP	wh-pronoun	<i>what, who</i>
NN	noun, sing. or mass	<i>llama</i>	WP\$	possessive wh-	<i>whose</i>
NNS	noun, plural	<i>llamas</i>	WRB	wh-adverb	<i>how, where</i>
NNP	proper noun, sing.	<i>IBM</i>	\$	dollar sign	\$
NNPS	proper noun, plural	<i>Carolinas</i>	#	pound sign	#
PDT	predeterminer	<i>all, both</i>	"	left quote	' or "
POS	possessive ending	<i>'s</i>	"	right quote	' or "
PRP	personal pronoun	<i>I, you, he</i>	(left parenthesis	[, {, <
PRP\$	possessive pronoun	<i>your; one's</i>)	right parenthesis],), }, >
RB	adverb	<i>quickly, never</i>	,	comma	,
RBR	adverb, comparative	<i>faster</i>	.	sentence-final punc	. ! ?
RBS	adverb, superlative	<i>fastest</i>	:	mid-sentence punc	: ; ... - -
RP	particle	<i>up, off</i>			

J&M Fig 5.6: Penn Treebank POS tags

Morphologically rich languages will often have 'morphosyntactic' tags = detailed breakdown of how the word parts combine, such as 'Noun+A3sg+P2sg+Nom' with possibly thousands of possibilities

Universal dependencies: 17 tags to try to catch phenomena that are distinct across all languages (there is also a 12 tag variant):

ADJ	adjective	NUM	numeral
ADP	adposition	PART	particle
ADV	adverb	PRON	pronoun
AUX	auxiliary	PROPN	proper noun
CCONJ	coordinating conjunction	PUNCT	punctuation
DET	determiner	SCONJ	subordinating conjunction
INTJ	interjection	SYM	symbol
NOUN	noun	VERB	verb
		X	other

13.0.2 Why is it hard?

Why is it always hard? Ambiguity.

glass of water/NOUN	vs	water/VERB the plants
lie/ VERB down	vs	tell a lie/NOUN
wind/ VERB down	vs	a mighty wind/NOUN (note these last are homographs)

time	flies	like	an	arrow
NOUN	VERB	MODAL	DET	NOUN
VERB	NOUN			
ADJ	NOUN	VERB		

What knowledge do we need?

1) A component deciding based on the word itself (some words only nouns, like arrow, some words ambiguous, a priori tag probability)

2) A component deciding based on the tags of surrounding words (a sequence of two determiners is rare, as is two base form verbs. Determiner almost always followed by adjective or nouns

Could we just put this into a linear model, predicting one tag at a time?

13.0.3 Why do we care?

This is the first step toward full-sentence syntax (structure trees). POS tags can also be used as input to tasks people do care about (sentiment analysis, word sense disambiguation).

In neural network land, could these be inferred automatically as ‘features’? With sufficient data, it seems they can, yes. But often there isn’t enough data, and linguistic intuition guides the search space, so that we can start off with prior knowledge of meaningful relationships between words.

More importantly, beyond text classification, this is a *sequence labeling* task, and the approaches we learn here, are specifically helpful for this and other sequence labeling tasks, of which here are two:

- Named Entity Recognition (NER): label words as beginning to persons (PER), organizations (ORG), locations (LOC), or none of the above: Barack/PER Obama/PER spoke/N from/N the/N White/LOC House/LOC today/N ./N
- Information field segmentation: Given specific text type (e.g. classified ad), find which words belong to which “fields” for db creation (price/size/location, author/title/year): 3BR/SIZE apt/TYPE in/N West/LOC Adams/LOC ,/N near/LOC USC/LOC ./N Bright/FEAT ,/N well/FEAT maintained/FEAT ...

Key features of sequence labeling: Correct label depends on

- item to be labeled (NER: Smith is probably a person. POS: chair is probably a noun)
- labels of surrounding items (NER: if following word is an organization (e.g. Corp.), then this word is more likely to be an organization. POS: if preceding word is a modal verb (e.g. will), then this word is more likely to be a verb)

The Hidden Markov Model (HMM) combines these sources probabilistically.

13.0.4 Bayes’ Law and Assumptions again

For word sequence $W = w_1, \dots, w_n$ we want the most likely tag sequence $T = t_1, \dots, t_n$, i.e.

$$\text{argmax}_T P(T|W) = \text{argmax}_T P(W|T)P(T)$$

Then we make some simplifying assumptions:

$$1) P(W|T) = P(w_1, \dots, w_n | t_1, \dots, t_n) = P(w_1 | w_2, \dots, w_n, t_1, \dots, t_n)P(w_2 | \dots) \dots P(w_n | t_1, \dots, t_n)$$

$$\approx P(w_1|t_1)P(w_2|t_2)\dots P(w_n|t_n)$$

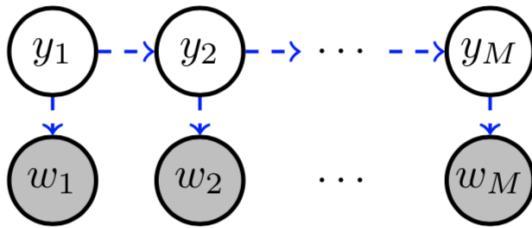
In other words we assume independence of all words and tags except w_i and t_i .

$$2) P(T) = P(t_1, \dots, t_n) = P(t_n|t_{n-1}, \dots, t_1)P(t_{n-1}|t_{n-2}, \dots, t_1)$$

$$\approx P(t_n|t_{n-1})P(t_{n-1}|t_{n-2})\dots P(t_2|t_1)P(t_1)$$

In other words we assume a tag is conditioned only on the previous tag. This is called the ‘Markov’ assumption.

We call $P(W|T)$ the *emission probability* and $P(T)$ the *transition probability*. A plate diagram helps (this is fig. 7.2 of Eisenstein):



Small implementation note: there’s no specific conditioning on the beginning or ending of a sentence, but it does seem like a good kind of probability to have. So in practice we can imagine sentences all beginning with, e.g. ‘BOS’ and ending with ‘EOS’. Note the probability of the first tag is $P(t_1)$ with no conditioning, but since the first tag is always the same (and the same word is always drawn from it) we don’t need to include those probabilities. It is, however, helpful to know $P(t_2|BOS)$ and to include a proper draw for ending, i.e. $P(EOS|t_n)$.

These probabilities are generally learned empirically and then smoothed (it’s much more important to smooth the emission probabilities...why?). Here are tables of empirically learned probabilities (these are from Jurafsky and Martin):

$t_{i-1} \setminus t_i$	NNP	MD	VB	JJ	NN	...
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	...
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	...
MD	0.0008	0.0002	0.7968	0.0005	0.0008	...
VB	0.0322	0.0005	0.0050	0.0837	0.0615	...
JJ	0.0306	0.0004	0.0001	0.0733	0.4509	...
...

$t_i \setminus w_i$	Janet	will	back	the	...
NNP	0.000032	0	0	0.000048	...
MD	0	0.308431	0	0	...
VB	0	0.000028	0.000672	0	...
DT	0	0	0	0.506099	...
...

So here are the probabilities that would be used to calculate the probability of the tagged sentence Time/NOUN flies/VERB like/MODAL an/DET arrow/NOUN:

$$\begin{array}{ccccccccc}
 \text{BOS} & \text{Time} & \text{flies} & \text{like} & \text{an} & \text{arrow} & \text{EOS} \\
 P(N|BOS) & P(V|N) & P(M|V) & P(D|M) & P(N|D) & P(EOS|N) \\
 P(\text{time}|N) & P(\text{flies}|V) & P(\text{like}|M) & P(\text{an}|D) & P(\text{arrow}|N)
 \end{array}$$

13.0.5 Viterbi Search

Things become tricky when trying to tag a new sequence; we want, remember, $\text{argmax}_T P(W, T)$ but finding this requires searching over the exponential number of sequences; even for a 5

word sequence over the 17-tag UD set, that's $17^5 = 1,419,857$ sequences. We turn instead to the Viterbi algorithm, a dynamic programming algorithm which uses the following intuition:

If we are at word $n - 1$ and have already calculated the best tag sequence ending in each of the 17 tag types at word $n - 1$, then the best tag sequence to word n (which has tag EOS) is one of 17 options: the best tag sequence to word $n - 1$ ending in tag 1 times the score for tag 1 to EOS, etc.

But the best tag sequence to tag 1 at word $n - 1$ is one of 17 options: the best tag sequence to $n - 2$ ending in tag 1 times the score for transition from tag 1 to tag 1 (times the emission for word $n - 1$ with tag 1), etc.

To make things complete, we assume all sequences start with a special BOS word that has an emission probability of 1 for the BOS tag

It's helpful to work through the following chart using the provided statistics; we'll do this in class (note that in code you probably want to add logs instead of multiplying probs):

```
for t in range(tagset):
    score[t][0] = emis[BOS][t]
for i, w in enumerate(words[1:]):
    for t in range(tagset):
        score[t][i] = max(score[:, i-1]*trans[t, :]*emis[w][t])
```

trans	N	V	D	P	A	EOS	emis	BOS	a	cat	doctor	in	is	the	very
BOS	.3	.1	.3	.2	.1	0	BOS	1	0	0	0	0	0	0	
N	.2	.4	.01	.3	.04	.05	N	0	0	.5	.4	0	.1	0	
V	.3	.05	.3	.2	.1	.05	V	0	0	0	.1	0	.9	0	
D	.9	.01	.01	.01	.07	0	D	0	.3	0	0	0	0	.7	
P	.4	.05	.4	.1	.05	0	P	0	0	0	0	1	0	0	
A	.1	.5	.1	.1	.1	.1	A	0	0	0	0	.1	0	0	

v	w0=BOS	w1=the	w2=doctor	w3=is	w4=in	EOS
BOS	1	0	0	0	0	0
EOS	0	0	0	0	0	.000027216
N	0	0	.0756	.001512	0	0
V	0	0	.00021	.027216	0	0
D	0	.21	0	0	0	0
P	0	0	0	0	.0054432	0
A	0	0	0	0	.00027216	0

13.0.6 OK, but what about more interesting features and discriminative models?

In fact, we can use the perceptron algorithm here, just as we used it for simple whole-text label prediction. Let's revisit perceptron:

```
def train(labeled_sentences, options, featsize):
```

```

# should be a better way to determine feat size
# probably want to initialize differently
model = {'theta': np.random(featsize)}
for i in range(iterations): # user-determined
    for sentence, label in labeled_sentences:
        hyp = classify(sentence, options, model)
        if hyp != label:
            model['theta'] += features(sentence, label) - features(sentence, hyp)
return model

def classify(sentence, options, model):
    scores = {}
    for option in options:
        scores[option] = evaluate(sentence, option, model)
    return max(scores.items(), key=operator.itemgetter(1))[0]

```

The real problem is the `classify` function, which I included here, or actually the `options` set. The point of `classify` is to find the maximum scoring sequence over all options. In sentiment classification there were only three sentiments, so you could just do 3 lookups and choose the best. But if there are t possible tags and the sequence is n words long, there are t^k possible sequences. Thankfully, dynamic programming in general (and the Viterbi algorithm specifically) lets us efficiently find the maximum score sequence. This then becomes what is known as *structured perceptron* since you are ultimately learning to predict the structured sequence of labels, and not just a single label at a time. We are no longer constrained to the emission and transition probabilities.

Here is the viterbi algorithm, rewritten to use general features, assuming an appropriate current θ and a `feats` function which returns features based on the last label t' , the current word w , and the proposed current label t (column vector notation won't work here but the complexity hasn't increased despite the deeper nesting):

```

for t in range(tagset):
    score[t][0] = theta*feats(emptyset, BOS, t)
for i, w in enumerate(words[1:]):
    for t in range(tagset):
        score[t][i] = -infinity
        for s in range(tagset):
            score[t][i] = max(score[t][i], score[s][i-1]*theta*feats(s, w, t))

```

What about logistic regression? Yes, this can be done as well (logistic regression over structures is called *conditional random fields* and the specific dependencies we allow make this a *linear chain conditional random field*), but it again requires efficiency changes, now to both inference and learning. For inference we have already seen how to efficiently calculate; the Viterbi algorithm is used to determine the argmax sequence. Because of the probabilistic nature of logistic regression, we need to calculate the sum of all scores (the denominator of softmax); this is an important component in gradient calculation. This can be calculated with a slight variant to the Viterbi algorithm called the *Forward algorithm* and its counterpart, the *Backward algorithm*.

Compare the Forward algorithm to the previous Viterbi algorithm:

```

for t in range(tagset):
    score[t][0] = theta*feats(\emptyset, BOS, t)
for i, w in enumerate(words[1:]):
    for t in range(tagset):
        score[t][i] = 0
        for s in range(tagset):
            score[t][i] += score[s][i-1]*theta*feats(s, w, t)

```

This calculates the partial score of **all** sequence labels from word 0 to t . The Backward algorithm does the same calculation, but starts at the end of the sequence:

```

for t in range(tagset):
    score[t][n] = 1
for i, w in reversed(enumerate(words)):
    for s in range(tagset):
        score[s][i] = 0
        for t in range(tagset):
            score[s][i] += score[t][i+1]*theta*feats(s, w, t)

```

Rather than directly calculate features, naturally, we can use neural networks, and this is in practice what is done today. However, the issue of exponential search still applies. Features are calculated via a bidirectional RNN (see chapter 8) and those features are provided to a CRF. The whole thing is differentiable and can be learned jointly (and is implemented using frameworks like PyTorch). The set of features for each possibility is also calculatable via the same dynamic programming approach used to calculate Forward/Viterbi.

Chapter 14

Constituency Parsing

14.0.1 First a little formal language theory

From a formal perspective a language is a (possibly infinite) set of strings (or sentences), where each string is made up of words from an alphabet (or vocabulary). This definition is amenable beyond human languages and has been applied to e.g. computer languages, DNA chains, and mathematical sequences. Much of formal language theory is concerned with the formal devices used to represent these languages.

14.0.2 Finite State Automata and Transducers

A weighted finite-state automaton (wFSA) is one such device. It is a 5-tuple $M = (Q, \Sigma, \lambda, \rho, \delta)$ where $Q = \{q_1, q_2, \dots, q_m\}$ is a finite set of states, Σ is a finite input alphabet of symbols, $\lambda : Q \rightarrow \mathbb{R}$ is an initial weight function, $\rho : Q \rightarrow \mathbb{R}$ is a final weight function, and $\delta : Q \times \Sigma \times Q \rightarrow \mathbb{R}$ is a transition function. Multiplication of λ for the chosen start state, iterated applications of δ , and ρ for the ending state yield the weight of the string formed by concatenating the series of Σ elements produced in the repeated applications of δ . FSAs and their unweighted counterparts have all kinds of nice properties such as closure under union, concatenation, intersection, and efficient best path, and are the mechanisms driving regular expressions.

Here are two simple wFSAs: the first recognizes any sequence of ‘a’ or ‘b’ that is of even length (for simplicity’s sake, the weights of each member of λ , ρ , and δ can be said to be 1 if the item appears, and 0 if it does not):

$$Q_2 = \{q_1, q_2\}; \Sigma = \{a, b\}, \lambda = \{q_1\}, \rho = \{q_1\}, \delta = \{(q_1, a, q_2), (q_1, b, q_2), (q_2, a, q_1), (q_2, b, q_1)\}$$

The second recognizes any sequence of ‘a’ that is of length divisible by 3:

$$Q_3 = \{q_7, q_8, q_9\}; \Sigma = \{a\}, \lambda = \{q_7\}, \rho = \{q_7\}, \delta = \{(q_7, a, q_8), (q_8, a, q_9), (q_9, a, q_7)\}$$

Their intersection recognizes any sequence of ‘a’ that is of length divisible by 6. Intersecting with a simple wFSA that recognizes exactly one string will yield either a wFSA with no transitions, indicating the string is not in the language, or a wFSA that returns that one string, indicating it is in the language. If the wFSA is probabilistic, then the string will be recognized with the probability it occurs in the language.

Finite-state machines can be used to e.g. represent HMMs, though to show this it is better to introduce a generalization, weighted finite state transducers (wFST):

A weighted finite-state transducer is a 6-tuple $M = (Q, \Sigma, \Omega, \lambda, \rho, \delta)$ where Ω is a finite alphabet of output symbols, $\delta : Q \times \Sigma \cup \{\epsilon\} \times \Omega \cup \{\epsilon\} \times Q \rightarrow \mathbb{R}$ is the revised transition function, and all other items are as before. ϵ is a special alphabet symbol which indicates no symbol is read/written. wFSTs are closed under composition, so if you have wFSTs M_1 and M_2 , you can build M_3 which behaves as $M_1(M_2)$, i.e. passing a string through the chain of transducers. This allows for the bigram HMM to be fairly cleanly written:

$$\begin{aligned}
M_{\text{trans}} &= \\
Q &= \{q_x \forall x \in \Sigma \cup \{q_{\text{START}}, q_{\text{END}}\}\} \\
\Sigma &= \{\text{Penn Treebank POS tags}\} \\
\Delta &= \Sigma \\
\lambda &= \{q_{\text{START}} \rightarrow 1\} \\
\rho &= \{q_{\text{END}} \rightarrow 1\} \\
\delta &= \{(q_i, j, j, q_j, P(j|i)) \forall i, j \in \Sigma \times \Sigma\} \cup \\
&\quad \{(q_{\text{START}}, i, i, q_i, P(i|\text{START})) \forall i \in \Sigma\} \cup \\
&\quad \{(q_i, \epsilon, \epsilon, q_{\text{END}}, P(\text{END}|i)) \forall i \in \Sigma\}
\end{aligned}$$

$$\begin{aligned}
M_{\text{emit}} &= \\
Q &= \{q\} \\
\Sigma &= \{\text{Penn Treebank POS tags}\} \\
\Delta &= \{\text{Vocabulary}\} \\
\lambda = \rho &= \{q \rightarrow 1\} \\
\delta &= \{(q, i, j, q, P(j|i)) \forall i, j \in \Sigma \times \Delta\}
\end{aligned}$$

14.0.3 Pushdown automata and context-free grammars

Some languages cannot be recognized by FSAs. For instance, the language $a^n b^n$, i.e. n ‘a’ followed by n ‘b’, for arbitrary n . More practically, the language consisting of strings with balanced parentheses (and possibly other symbols) is also not recognizable by FSAs. The proof of this is quite elegant (it involves something called the ‘pumping lemma’). This is particularly troubling to us because we would like to bracket sentences into hierarchical syntactic chunks.

For example,

$[[\text{We}/\text{PRP}]_{NP} [\text{would}/\text{MD} [\text{like}/\text{VB} [[\text{to}/\text{TO} [\text{bracket}/\text{VB} [\text{sentences}/\text{NNS}]_{NP} [\text{into}/\text{IN} \dots]]_{S}]]_{VP}]]_{VP}]]_{VP} \dots]_S$

This is somewhat more elegantly (but less compactly) written as in Figure 14.1.

A mechanism that can represent such languages is the *weighted pushdown automaton*, which is like an FSA but with a stack. However it’s far more common to see these languages

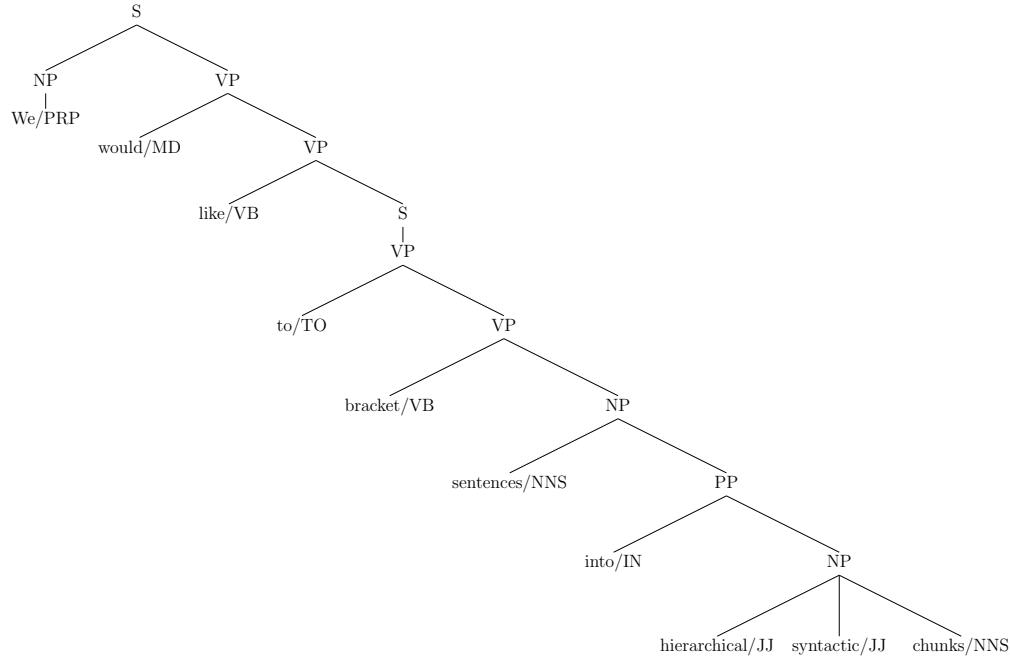


Figure 14.1: Syntactic Tree

written in an equivalent formalism, the *weighted context-free grammar* (wCFG). This is a 4-tuple (N, Σ, R, S) where N is a set of non-terminal symbols, σ is a set of terminal symbols, $S \in N$ is a designated start symbol, and $R : N \times (N \cup \Sigma)^* \rightarrow \mathbb{R}$ are the production rules. Here is a CFG (weights of productions shown assumed to be 1) for $a^n b^n$ ($N = \{S\}$, $\Sigma = \{a, b\}$):

$$S \rightarrow ab$$

$$S \rightarrow aSb$$

Here are the productions that can be used to form the tree in Figure 14.1 (excluding tags for POS to word; POS listed in lowercase):

$$\begin{aligned}
S &\rightarrow NP VP. \\
S &\rightarrow VP \\
NP &\rightarrow prp \\
NP &\rightarrow nns PP \\
NP &\rightarrow jj jj nn \\
VP &\rightarrow md VP \\
VP &\rightarrow vb S \\
VP &\rightarrow to VP \\
VP &\rightarrow vb NP \\
PP &\rightarrow in NP
\end{aligned}$$

14.0.4 Penn Treebank

Built in the 90s at Penn, for about 1 million dollars. 1 million words in about 40,000 sentences of WSJ text. First large scale analysis of naturally occurring syntax (other components include much more POS tagging, speech annotation). Compared to the POS tag set, there are many fewer tree labels:

Table 1.2. The Penn Treebank syntactic tagset

ADJP	Adjective phrase
ADVP	Adverb phrase
NP	Noun phrase
PP	Prepositional phrase
S	Simple declarative clause
SBAR	Subordinate clause
SBARQ	Direct question introduced by <i>wh</i> -element
SINV	Declarative sentence with subject-aux inversion
SQ	Yes/no questions and subconstituent of SBARQ excluding <i>wh</i> -element
VP	Verb phrase
WHADVP	Wh-adverb phrase
WHNP	Wh-noun phrase
WHPP	Wh-prepositional phrase
X	Constituent of unknown or uncertain category
*	“Understood” subject of infinitive or imperative
0	Zero variant of <i>that</i> in subordinate clauses
T	Trace of wh-Constituent

However, the annotation guide for the treebank is 318 pages long (compared to 37 for POS tags).

This was meant to be a documentation of how people really constructed (English, largely news) sentences rather than being told what was and was not done by linguists. Also now we could think, if given a new sentence, can we automatically annotate it in the same way?

BTW, the treebank is divided into sections, and it's common for people to train on sections 2-21, use dev for section 22, and evaluate on section 23. This has enabled comparable

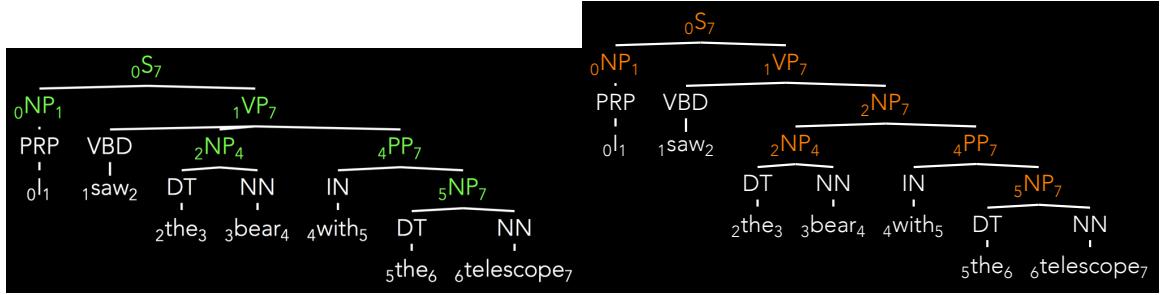


Figure 14.2: Two parses of a sentence

results over about 25 years, but there is concern that we've overfit on this data set by now! There are other treebanks that have been constructed, for other languages, and for English, but none is as consistent or large as Penn Treebank.

14.0.5 Evaluation

Parsing scores are typically given as an F1 measure, comparing gold (i.e. reference) to hypothesis brackets. Let's assume the left side of Figure 14.2 is the gold sentence and the right side is the hypothesis. Then, the brackets for each are:

gold	hyp
(S 0 7)	(S 0 7)
(NP 0 1)	(NP 0 1)
(VP 1 7)	(VP 1 7)
(NP 2 4)	(NP 2 7)
(PP 4 7)	(NP 2 4)
(NP 5 7)	(PP 4 7)
	(NP 5 7)

F1 is the harmonic mean of *recall* and *precision*. In such a scenario we can divide up items in a response into *correct* (items in hypothesis and reference), *missed* (items in reference but not hypothesis), and *spurious* (items in hypothesis but not reference). *Precision* is $\frac{\text{correct}}{\text{correct} + \text{spurious}}$. *Recall* is $\frac{\text{correct}}{\text{correct} + \text{missed}}$. $F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$.

In the example above, 6 items are correct, one is spurious, and none are missed. Thus the precision is $6/7 = .857$, the recall is $6/6 = 1.0$, and the F1 is $.923$.

Given a grammar and a sentence, how do we efficiently find a parse tree for that sentence? More importantly, how do we find the most likely parse tree? The core of those answers are in the CKY algorithm, a bottom-up dynamic programming algorithm. CKY requires rules be in a special form called ‘Chomsky Normal Form’ (CNF) that only allows rules of the following form:

$$X \rightarrow Y Z$$

$$X \rightarrow a$$

where X, Y, Z are nonterminals and a is terminal. There must be *exactly* two nonterminals on the RHS of a rule or there can be exactly one terminal. Although there are well established techniques for converting, for now let's assume we already have a grammar in this form.

Here's the core algorithm (see also alg. 13 on p. 241 of eisenstein):

```
chart = defaultdict(lambda: defaultdict(set)) # start->end->labels
bpt = defaultdict(lambda: defaultdict(set)) # start->end->backpointers
for w in range(1, len(sent)+1): # width
    for start in range(len(sent)-w+1):
        end = start+w
        if w == 1:
            for lhs in terms[words[start]]:
                chart[start][end].add(lhs)
        else:
            for mid in range(start+1, end):
                for rhs1 in chart[start][mid]:
                    for rhs2 in chart[mid][end]:
                        for lhs in nterms[rhs1][rhs2]:
                            chart[start][end].add(lhs)
                            bpt[start][end].add((mid, rhs1, rhs2))
```

Let's work it out with a small example:

time	flies	like	an	arrow
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
S→NP VP				
NP→DT NN	[1,2]	[1,3]	[1,4]	[1,5]
NP→time fruit				
NP→NN NNS				
VP→ VBP NP		[2,3]	[2,4]	[2,5]
VP→ flies				
VP→ VP PP			[3,4]	[3,5]
PP→ IN NP				
DT→ a an				
NN→time fruit arrow banana				[4,5]
NNS→flies				
VBP→ like				
IN→ like				

When done you should have this:

time	flies	like	an	arrow
NP [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN [2,3] VBP	[2,4]	PP [2,5] VP
			DT [3,4]	NP [3,5]
				NN [4,5]

```

graph TD
    S --- NP1[ ]
    S --- VP1[ ]
    NP1 --- time[time]
    NP1 --- VP2[ ]
    VP2 --- VP3[ ]
    VP2 --- PP1[ ]
    VP3 --- flies[flies]
    VP3 --- IN1[ ]
    IN1 --- like[like]
    PP1 --- DT1[ ]
    PP1 --- NN1[ ]
    DT1 --- an[an]
    NN1 --- arrow[arrow]
  
```

14.0.6 Building probabilistic grammars

How do we actually get grammars? We can read them off of the trees in the treebank. As seen above, these are not always in CNF. Simpler than changing the grammar, we can modify the trees themselves. We can do the same with unary chains, collapsing them.

How to choose probabilities for grammar rules? To be probabilistic, the sum of all rules with the same LHS should be 1.0; given that we're still being generative here, this is $P(T|W)P(T) = P(T, W)$ and with the chain rule and markov and independence assumptions as before, we ultimately want a set of $P(RHS|LHS)$ probabilities to multiply. We calculate these empirically from the corpus. We then can modify our CKY algorithm above to calculate weights and choose the maximum for every LHS in a cell.

Here is the example from above with weights to work through:

time	flies	like	an	arrow
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
$S \rightarrow NP VP^{1.0}$				
$NP \rightarrow DT NN^{0.3}$	[1,2]	[1,3]	[1,4]	[1,5]
$NP \rightarrow time fruit^{0.05}$				
$NP \rightarrow NN NNS^{0.6}$				
$VP \rightarrow VBP NP^{0.7}$		[2,3]	[2,4]	[2,5]
$VP \rightarrow flies^{0.1}$				
$VP \rightarrow VP PP^{0.2}$				
$PP \rightarrow IN NP^{1.0}$			[3,4]	[3,5]
$DT \rightarrow a an^{0.5}$				
$NN \rightarrow time fruit arrow banana^{0.25}$				
$NNS \rightarrow flies^{1.0}$				
$VBP \rightarrow like^{1.0}$				
$IN \rightarrow like^{1.0}$				

And here it is filled out:

time	flies	like	an	arrow
NP [0,1]	S NP [0,2]	IN VBP [2,3]	DT [3,4]	NN [4,5]
$NP \rightarrow 0.05$	$S \rightarrow 0.005$			0.0039375
$NN \rightarrow 0.25$	$NP \rightarrow 0.15$			$VP \rightarrow 0.00075$
$NP \rightarrow DT NN^{1.0}$	$VP \rightarrow 0.1$			$PP \rightarrow 0.0375$
$NP \rightarrow time fruit^{0.05}$	$NN \rightarrow 1.0$	$IN \rightarrow 1.0$		$VP \rightarrow 0.02625$
$NP \rightarrow NN NNS^{0.6}$		$VBP \rightarrow 1.0$		
$VP \rightarrow VBP NP^{0.7}$		$[2,3]$	$[2,4]$	$[2,5]$
$VP \rightarrow flies^{0.1}$				
$VP \rightarrow VP PP^{0.2}$				
$PP \rightarrow IN NP^{1.0}$				
$DT \rightarrow a an^{0.5}$				
$NN \rightarrow time fruit arrow banana^{0.25}$				
$NNS \rightarrow flies^{1.0}$				
$VBP \rightarrow like^{1.0}$				
$IN \rightarrow like^{1.0}$				

The analogue to the forward algorithm, the *inside* algorithm, computes the partition function (sum of probabilities of all trees) by replacing max with add.

Next time: head lexicalization, dependency parsing.

Chapter 15

Constituency Parsing Continued

Last time we went through the mechanics of using CKY and viterbi to find a parse tree and simple maximum likelihood to get probabilities for the grammar components.

Turns out this gets us parse scores of around 73% which is not good; modern parsers are in the mid-90s. Why? Both because the rules extracted from trees are too specific and because they're not specific enough!

15.0.1 Too Specific – Markov binarization

Consider the tree:

```
(NP
  (DT the)
  (JJS tallest)
  (NN steel)
  (NN building)
  (NN antenna)
  (PP
    (IN in)
    (NP
      (NNP America))))
```

This yields the rule

```
NP -> DT JJS NN NN NN PP
```

which is totally useless for

```
(NP
  (DT the)
  (JJS tallest)
  (NN building)
  (PP
    (IN in)
    (NP
      (NNP America))))
```

Previously I (hopefully in class, but not in notes) mentioned that changing the trees is an easy way to get into CNF. For perfect reconstruction you can do this by introducing one-time nonterminals, e.g.

```
(NP
  (DT the)
  (T145-1
    (JJS tallest)
    (T145-2
      (NN steel)
      (T145-3
        (NN building)
        (T145-4
          (NN antenna)
          (PP
            (IN in)
            (NP_NNP America)))))))
```

Which you could also label by the pattern of the RHS being replaced.

```
(NP
  (DT the)
  (DT^JJS^NN^NN^NN-1
    (JJS tallest)
    (DT^JJS^NN^NN^NN-2
      (NN steel)
      (DT^JJS^NN^NN^NN-3
        (NN building)
        (DT^JJS^NN^NN^NN-4
          (NN antenna)
          (PP
            (IN in)
            (NP_NNP America)))))))
```

But these aren't terribly flexible. If we introduce *reusable* nonterminals we can get more flexible rules. For instance, we can remember just the nonterminal we're in and the pos tag to our left:

```
(NP
  (DT the)
  (NP^DT
    (JJS tallest)
    (NP^JJS
      (NN steel)
      (NP^NN
        (NN building)))
```

```
(NP^NN
  (NN antenna)
  (PP
    (IN in)
    (NP_NNP America))))))
```

This gives us rules like

$NP^JJS \rightarrow NN\ NP^NN$

which are useful in the smaller sentence.

15.0.2 Not Specific Enough I: modeling parent-child behavior

Let's say we saw:

```
(NP
  (NP the man)
  (PP in the car))
```

90 times (eliding the unimportant details of the trees)
and

```
(NP
  (NP the man)
  (PP in the car)
  (PP with the dog))
```

10 times. Using MLE, we get

```
NP → NP PP 90/200 = .45
NP → NP PP PP 10/20 = .05
(NP → DT NN 100/20 = .5)
```

What happens if we then want to parse ‘the man in the car with the dog’? The sentence seen in training scores .05 for the combining rule, while this parse:

```
(NP
  (NP
    (NP the man)
    (PP in the car))
  (PP with the dog))
```

scores $.45 \times .45 = .2020$ for the combining rules! This shouldn't be! A tree not seen in training scores higher than the exact tree seen in training!

What can we do? Annotate with more context – give nonterminals their parent symbols as well:

```
(NP^VP
  (NP^NP the man)
  (PP^NP in the car))
```

and

```
(NP^VP
  (NP^NP the man)
  (PP^NP in the car)
  (PP^NP with the dog))
```

Now the rule

$\text{NP}^{\wedge}\text{VP} \rightarrow \text{NP}^{\wedge}\text{NP } \text{PP}^{\wedge}\text{NP}$

can't be used twice!

15.0.3 Not Specific Enough II: modeling lexical behavior

It turns out words do matter! Consider:

```
(S
  (NNS workers)
  (VP
    (VP
      (VBD dumped)
      (NP
        (NNS sacks)))
    (PP
      (IN into)
      (NP
        (DT a)
        (NN bin))))
```

seems correct. The dumping is into a bin. Consider an alternative:

```
(S
  (NNS workers)
  (VP
    (VP
      (VBD dumped)
      (NP
        (NP
          (NNS sacks)))
      (PP
        (IN into)
        (NP
          (DT a)
          (NN bin))))
```

They dumped a thing called “sacks into a bag.” Seems wrong.
Which is more likely? It comes down to the difference between these two rules:

VP → VP PP

NP → NP PP

Both are good. Neither are *directly* compared. This seems arbitrary. But PPs with ‘into’ have a strong preference to be attached to VPs, not NPs. Consider if it was instead “tomatoes from the outbreak” (which all have the same POS tags as “sacks into a bag”). What to do? Annotate labels with their lexical ‘heads.’ What’s a head? the most important word in a phrase. How are these determined? Rules, actually. That were written down in 1995.¹ Here’s an example for VP: [VBD VBN MD VBZ TO VB VP VBG VBP ADJP NP]. Use the leftmost of the first of these categories, if it appears. If it doesn’t use the left most of the next one and so on. Following the rules you get:

```
(S/dumped
  (NNS/workers workers)
  (VP/dumped
    (VP/dumped
      (VBD dumped)
      (NP/sacks
        (NNS/sacks sacks)))
    (PP/into
      (IN/into into)
      (NP/bin
        (DT/a a)
        (NN/bin bin)))))
```

Now we are comparing these rules

```
VP/dumped → VP/dumped PP/into
NP/sacks → NP/sacks PP/into
```

The first seems much more likely.

(In an older version of this course we’d now have to talk about how there aren’t likely to be sufficient statistics to estimate these fairly fine-grained rules, so we’ll have to add smoothing. The smoothing for parse trees can be quite complicated and would take one lecture at least, but using neural approaches ends up allowing us to skip over all of that.)

Lexicalization proved very important! So important that there isn’t a lot of constituent parsing research any more since a new formalism for syntax became dominant...dependency trees!

¹<http://www.cs.columbia.edu/~mcollins/papers/heads>

Chapter 16

Dependency Parsing

16.1 Why dependencies?

It turns out *lexicalization* is pretty important to syntactic parsing. This is at first a bit surprising, since syntax is concerned with the order of words and not their content. The famous sentence ‘Colorless green ideas sleep furiously’ (Chomsky 1957) is an example of a syntactically valid but semantically vacant sentence. It clearly has a parse:

```
(S
  (NP
    (JJ colorless)
    (JJ green)
    (NN ideas))
  (VP
    (VBP sleep)
    (RB furiously)))
```

But as we previously saw, rules like $S \rightarrow NP\ VP$ without lexicalization can lead to lots of apparent ambiguity.

Another problem is that not all syntactic behavior occurs in contiguous phrases. This is particularly true in languages with *free word order* but even occurs in English; e.g. ‘The hearing is scheduled on the issue today.’ There is a relationship between ‘on the issue’ and ‘the hearing’ but that relationship isn’t really expressible with contiguous phrases only.

Finally, as sentence length grows a significant part of the parse is far from the word level and this, it turns out, is less helpful in downstream tasks. We mostly want to know the relationship between words in a sentence. Specifically the connection between phrases and the *heads* of sub-phrases contained within them is important.

What is a head? This is actually a tricky linguistic question. Informally it’s:

- The most important word in a phrase
- The main content word in a phrase
- The word that determines the phrase’s label

- The word that other phrases have to agree with (e.g. in case or gender)

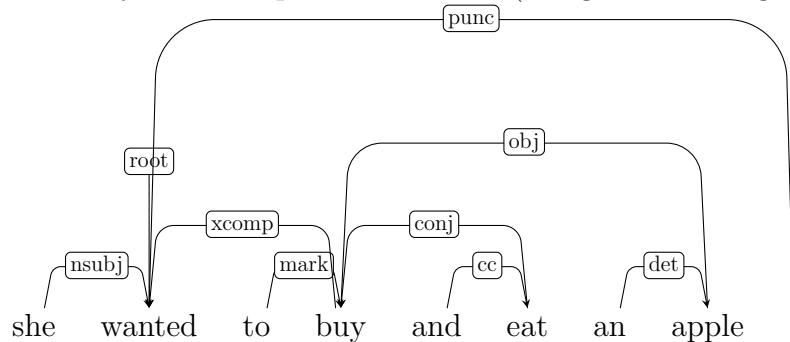
This can sometimes lead to inconsistencies. In the prepositional phrase ‘with the mayor’ is ‘with’ the head (determines the label) or is ‘mayor’ (the main content word)? It depends on which linguist you ask; the rules behind the Penn treebank say the preposition, while the rules behind universal dependencies say the noun. As long as you use consistent head finding you will be okay.

16.2 What are dependencies?

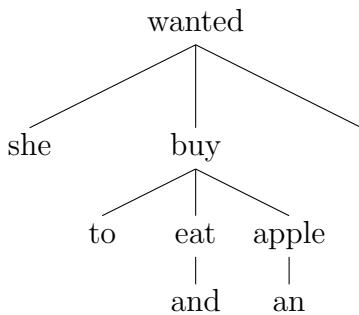
Here is an example for the sentence ‘She wanted to buy and eat an apple’ (from the UD guidelines):

1	she	2	nsubj
2	wanted	0	root
3	to	4	mark
4	buy	2	xcomp
5	and	6	cc
6	eat	4	conj
7	an	8	det
8	apple	4	obj
9	.	2	punct

Pictorially we can represent it like so (using the amazing tikz-dependency package):



Or this:



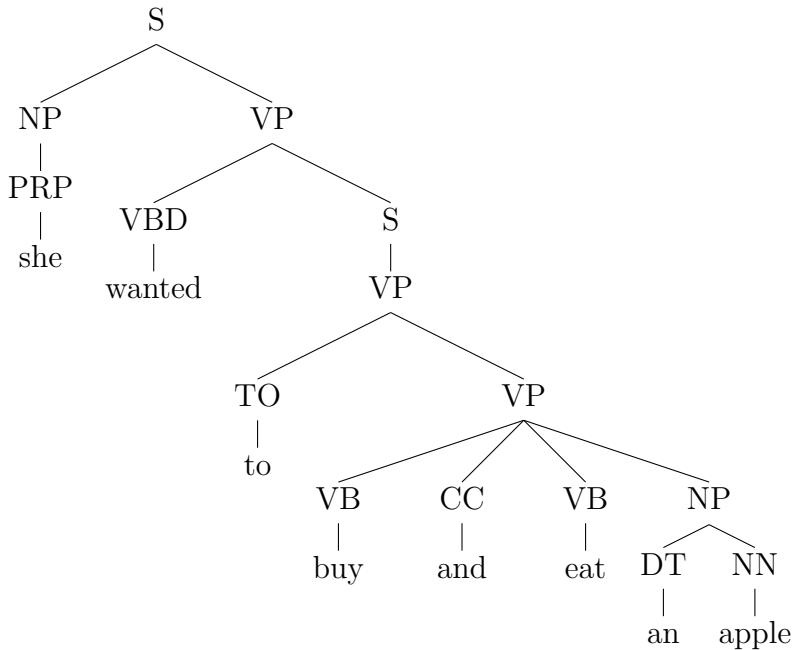
- Every word in a sentence except one has one parent (or governor) word, which is the head of the smallest syntactic unit it is not the head of. A word may have zero or more words that consider it their head.

- The word which has no head is the *root* of the sentence.
- Each parent-child relationship may be annotated with a label connoting the role of the phrase the child is the head of. There are 37 such label types in universal dependencies.

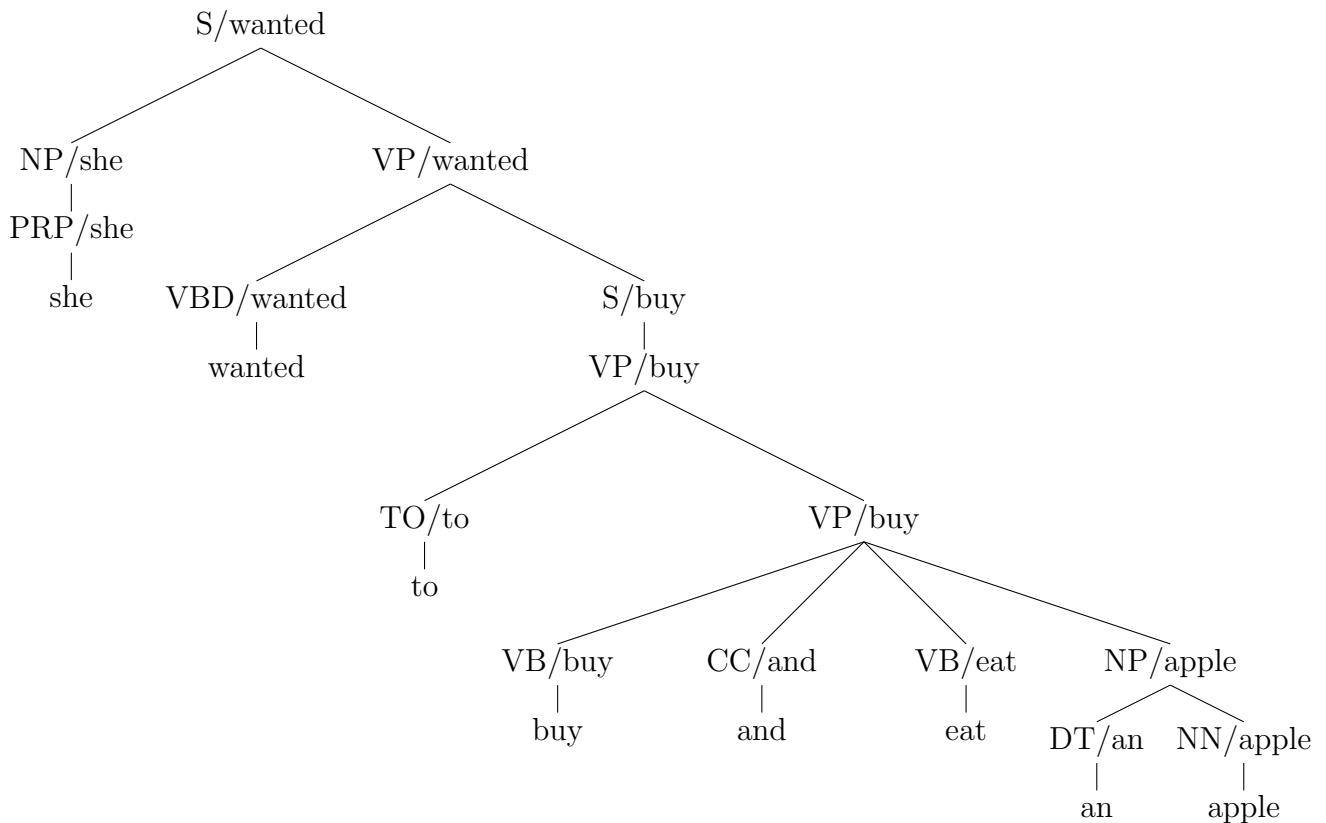
The consequences of these requirements (particularly the first two) is that this will form a tree. There is an extension to this formalism called *enhanced dependencies* that annotates more relationships and forms *graphs* (i.e. words can have more than one head). We won't discuss them here.

16.3 Conversion from Constituencies, planarity, and projectivity

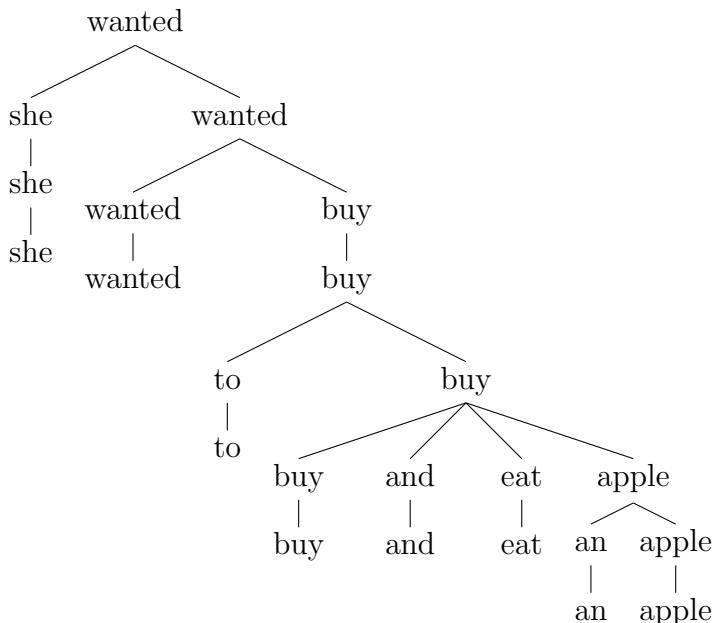
What is the relationship between constituencies and dependencies? You can convert a constituent tree into a (unlabeled) dependency tree as follows:



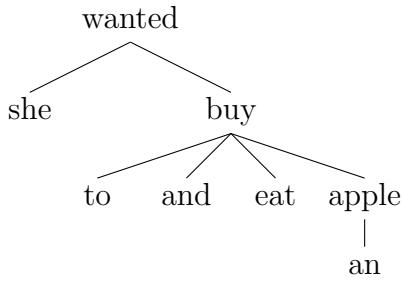
1. Head-lexicalize the tree



2. remove phrase labels

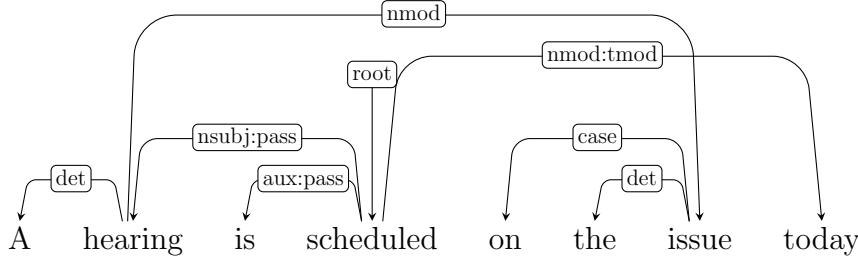


3. merge children into parents with same label



Notice it's not the same dependency as above! This is because the original example was annotated as a dependency and this is converted from another structure, with different annotation standards and possibly different head rules.

Converting from dependencies to constituencies is in general not possible. Apart from the labeling problem (which also exists in the other direction) there is too much ambiguity introduced in the simpler dependency structure. More importantly, dependencies cannot be converted at all if they are not *planar*¹, i.e. if the arcs cross when the words are arranged in order. Such a tree is *non-projective*. Here is an example:



Thankfully non-projective dependencies are pretty rare in English.

16.4 Parsing Methods

It turns out that parsing methods for dependencies are often a lot faster than those for constituencies. The one we will discuss, *shift-reduce*, is linear-time and greedy (though it can be beamed) and can take a wide variety of features. It doesn't handle non-projectivity, however. The second one, *Chiu-Liu-Edmonds*, is quadratic and optimal but somewhat limited in its feature set. We won't discuss it in detail but I provide several pointers.

16.4.1 Shift-Reduce

Big idea of shift-reduce parsing: you keep a *stack* of words/partial structures you are processing and a *buffer* of words you haven't started processing yet. At each time step you do some work (an *operation*) at the top of the stack. In 'arc-standard' parsing there are the following operations:

- SHIFT = move a word from the buffer to the top of the stack

¹misleading to those very familiar with graph theory; see Kuhlman 1998

- LEFT-ARC-*label* = top of stack is parent of second in stack; add *label*; top of stack stays in (pop 2nd)
- RIGHT-ARC-*label* = second in stack is parent of top in stack; add *label*; second in stack stays in (pop top)

So this becomes a classification problem with $1 + 2 \times k$ labels choices. We'll get into what makes good features for this but it's first helpful to walk through a parse. Additionally, in order to train a classifier you need a lot of examples of a configuration and a choice of label. There is a general procedure for converting from a dependency tree into the sequence of parse steps that will form it. Using the tree as a guide:

- If `stack[0]` is the parent of `stack[1]` with label *l*, LEFT-ARC-*l*.
- If `stack[1]` is the parent of `stack[0]` with label *l* and no dependents of `stack[0]` are still in the buffer, RIGHT-ARC-*l*.
- Otherwise, SHIFT

Here is an example parse tree (unlabeled) and a walkthrough; we'll go over it in class:

	ROOT ROOT book ROOT book the ROOT book the flight ROOT book flight ROOT book flight through ROOT book flight through houston ROOT book flight houston ROOT book flight ROOT book ROOT	book the flight through houston the flight through houston flight through houston through houston through houston houston	SHIFT SHIFT SHIFT LEFT SHIFT SHIFT LEFT RIGHT RIGHT RIGHT Done	the ← flight through ← houston flight → houston book → flight ROOT → book
--	---	--	--	---

A potential problem is that shift-reduce is *greedy* and an early bad decision can lead to later problems. We can beam, i.e. consider *k* possibilities simultaneously. We then consider the *k* best successors of these, trim back to only *k*, and continue. This is still linear in sentence length, i.e. nk^2 .

Another problem is that arc-standard is strictly ‘bottom-up’ in that it is cautious, particularly about RIGHT, e.g. waiting a long time after seeing the initial ‘book flight’ to make that arc. The useful features could be inaccessible to a classifier. A variant, called ‘arc-eager’ seeks to improve things. It has slightly different definitions and one more operation:

- SHIFT = (as before) move a word from the buffer to the top of the stack

- LEFT-ARC-*label* = top of *buffer* is parent of top in stack; add *label*; top of stack is popped.
- RIGHT-ARC-*label* = top of stack is parent of top in buffer; add *label*; shift buffer to stack
- REDUCE = pop the top of the stack

The heuristics for converting from a tree to an instruction are simpler; if the top of the buffer is parent to the top of the stack, do LEFT; if the top of the stack is parent to the top of the buffer, do RIGHT; otherwise if you can SHIFT, do it, otherwise REDUCE. Here's how the parse goes under arc-eager:

ROOT	book the flight through houston	RIGHT	ROOT → book
ROOT book	the flight through houston	SHIFT	
ROOT book the	flight through houston	LEFT	the ← flight
ROOT book	flight through houston	RIGHT	book → flight
ROOT book flight	through houston	SHIFT	
ROOT book flight through	houston	LEFT	through ← houston
ROOT book flight	houston	RIGHT	flight → houston
ROOT book flight houston		REDUCE	
ROOT book flight		REDUCE	
ROOT book		REDUCE	
ROOT		Done	

Neural Dependency Parser (HW3)

An excellent application of neural networks to dependency parsing is the work of Danqi Chen (student of Manning, now professor at Princeton) from 2014 (ancient history!). It's pretty straight-forward and still uses hand-engineered features; the trick is that it uses a lot of them, and the neural network takes care of the smoothing. The features are:

- the first three words on the stack and the buffer (and their POS tags) (12 features)
- the words, POS tags, and arc labels of the first and second leftmost and rightmost children of the first two words on the stack, (24 features)
- the words, POS tags, and arc labels of leftmost child of the leftmost child and rightmost child of rightmost child of the first two words of the stack (12 features)

Collobert et al (2011) pretrained word embeddings were used; the rest were learned. A n^3 activation function is used (quite unusual nowadays but Chen was writing her code all by hand). The parser scored 90.7 LAS and 92.0 UAS on converted treebank, a SOTA for the time. It was also much faster than other parsers; this was in large part due to a lot of precomputation of values. The latest (2019) using BERT/XLNet is around 95.7 LAS and 97.2 UAS.

16.4.2 Chiu-Liu-Edmonds

Shift-reduce parsers have a major flaw; they can't handle non-projective trees. For English this isn't a problem; over 99.4% of the English (and 100% of the Chinese) Treebank is projective. But for Czech this would be a problem. Another algorithm, called Chiu-Liu-Edmonds, after its simultaneous creators, can be used instead. It's a fairly elegant algorithm but unless there's great demand I'll leave it to you to research (e.g. <https://www.cs.cmu.edu/~sswayam/talks/cle.pdf>, https://user.phil.hhu.de/~waszczuk/teaching/depparse-su18/exercises/session_5/example.pdf)

Chapter 17

Semantics

17.1 Semantics

This is potentially a huge topic – note that it takes up three chapters of Eisenstein. Semantics is the study of how to understand the meaning behind language. The question of what meaning even is gets a bit philosophical and this is part of why the topic is potentially large. We can think of semantics, or rather semantic analysis (per Eisenstein’s description) as converting natural language into a *meaning representation* that connects to knowledge about the world. Further, each thing that is known should have a single representation such that if the representation changes, the meaning changes.

We’re going to focus on some key parts of meaning. So to begin with, here’s what we’re not going to talk about:

- Logical semantics – the interpretation of natural language as a set of logical formulas including negation, conjunction, disjunction, implication, associativity, etc. You can read more about this in chapter 12 of Eisenstein.
- Semantic roles and predicate argument structure that are covered in 13.1 and 13.2, as well as abstract meaning representation, which uses some of these (see 13.3); I meant to talk about these but I will defer for now due to time.

Most of the focus will be on lexical semantics, first traditional and then distributional. By lexical semantics we mean the meaning of words.

17.2 Semantic Similarity

To begin with, instead of analyzing each word in myriad dimensions of potential meaning, let’s simply discuss the relative relationship of words to each other. If two words are interchangeable we can say they are ‘synonyms.’ Why is it useful to determine these? Consider a question answering task. We want to answer this:

What is a good way to remove wine stains?

We can use a few rules and search a big corpus for sentences starting ‘A good way to remove wine stains is.’ But we can do better. We’d like to match any of these as well:

Salt is a great way to eliminate wine stains
How to get rid of wine stains
How to get red wine out of clothes
Oxalic acid is infallible in removing iron-rust and ink stains
Knowing that ‘remove’ is synonymous or at least similar to ‘eliminate’, ‘get rid of’, ‘get out of’, ‘removing’ and that ‘good’ is similar to ‘great’ would help in approximate matching.
You could use this to find movie recommendations; find movie scripts with similar words (or phrases, sentences, paragraphs) to scripts of movies you like.

We’re pretty good at doing this. You probably don’t need to be told which of these pairs are similar and which are not similar:

bank-money	doctor-beer
apple-fruit	painting-January
tree-forest	money-river
bank-river	apple-penguin
pen-paper	nurse-fruit
run-walk	pen-river
mistake-error	clown-tramway
car-wheel	car-algebra

It turns out humans ranking similarity on a scale from 0 to 4 do so with correlation of 0.9! that’s pretty good!

What makes words similar?

- Meaning: (wait isn’t all semantics meaning?) e.g. ‘want’ vs ‘desire’
- World knowledge: things go together (pen-ink, dog-cat)
- psychology: we think of the concepts together (death-taxes)

17.2.1 Brief bit on linguistic terms

These definitions are sometimes helpful. You are probably familiar with some if not all of them from grade school.

- homonym: two words with same form but unrelated, distinct meanings.
 - homograph: bank (finance) vs bank (slope); bat (wood stick) vs bat (animal)
 - homophone: write vs right, piece vs peace
- polysemy: having more than one related meaning. bank (financial institution vs physical building)
- metonymy: one thing standing in for another (‘I love Jane Austen’(‘s writing)), which can lead to introduction of senses e.g. ‘school’ to mean the two bank senses
- synonym: two words with same meaning
- antonym: two words with opposite meaning

17.2.2 Evaluation

Some ways to evaluate:

- Given a word and a choice of other words, find the one that is closest in meaning.
 - **accidental**: wheedle, ferment, inadvertent, abominate
 - **imprison**: incarcerate, writhe, meander, inhibit
 - WS353: a dataset of similarity scores for 353 English word pairs. Can be used to automatically create these tests.
- Malapropism test: find the word in the sentence that is most likely wrong
 - Jack withdrew money from the ATM next to the band.
 - Can be created by randomly replacing words from a lexicon

17.2.3 Hand-Built Resources

We cared about annotating semantic relationships between words, so much so that considerable effort was put into hand-crafting ontologies. For lexical semantics, the most famous (other than roget's thesaurus) was **WordNet**. It has 118k English nouns, 11.5k verbs, 22.5k adjectives, 5k adverbs.

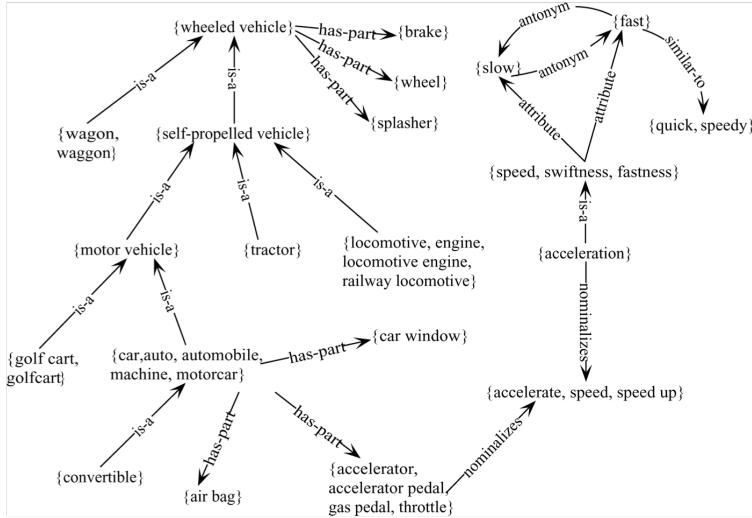
Lemmas (base word forms) have one or more senses (distinct meaning units), each with examples explaining the distinct meaning. Meaning units are known as *synsets*. A synset has a definition and often some examples. A lemma has multiple senses. Each sense has one synset. But different senses of different words can share the same synset.

E.g. chump-1 has the synset defined as “a person who is gullible and easy to take advantage of”. This synset is shared by fool-2, gull-1, mark-9, patsy-1, fall guy-1, sucker-1, soft touch-1, mug-2.

Senses are structured in hypernym trees; a hypernym is more inclusive, a hyponym less inclusive. *car* is a hyponym of *vehicle*. Alternately this is an ‘is-a’ hierarchy (*car* is a *vehicle*). Alternately it’s an entailment hierarchy (being a *car* entails being a *vehicle*)

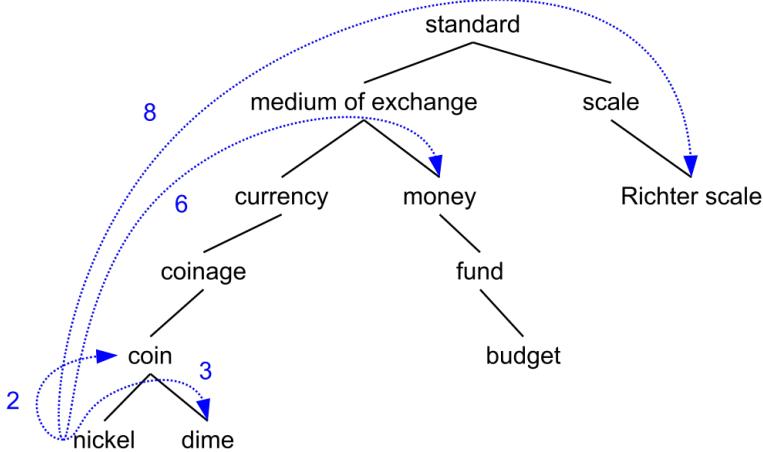
Wordnet also encodes *meronymy* (but less so); this is a part-whole relation; *leg* is a meronym of *chair*. Alternately, *chair* is a *holonym* of *leg*.

There are verb relations too but they’re more incomplete.



Wordnet is incomplete and inconsistent; some parts are very dense, others have gaping holes. It is also increasingly out-of-date (e.g. *television* has meronym *kinescope*—these haven't been part of televisions for years!). Nevertheless it can be a very precise (subject to datedness) repository of info and can be used to determine word similarity with some quite simple algorithms over synset trees.

E.g. Path length == number of arcs you walk to get between words. A simple normalized refinement is $simpath = 1/\text{pathlen}$. It's a number from 0 to 1 and the closer words are, the higher the value is.



17.2.4 Distributional Methods

A totally different way to understand word similarity is based on a famous quote by linguist John Rupert Firth (1957): “You shall know a word by the company it keeps.” That is, words are similar if the words they are near are similar.

Intuition from Zelig Harris (another linguist) in 1954: “oculist and eye-doctor occur in almost the same environments...thus we say they are synonyms.”

Here's another example:

A bottle of tesgüino is on the table

Everybody likes tesgüino
 Tesgüino makes you drunk
 We make tesgüino out of corn.

What do you think tesgüino is?

17.2.5 Word co-occurrence matrices and mutual information

Let's try this first comparing documents and words:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Notice the usage patterns of ‘fool’ and ‘clown’ vs ‘battle’ and ‘soldier.’ Notice also the similarity of some of the plays.

Important difference from thesaurus-based approaches now; we’re losing the ability to distinguish between senses of the same word form (there are ways to try to get these back but won’t cover them here and the ‘hard decision’ approach doesn’t work that well...but we will revisit this when we discuss contextualized representations).

We can also make such a table for smaller contexts, such as a four-word window.

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

In this table the number of times a word in the column was seen within four words of the word in the row is listed in the cell.

In reality this table is $|V| \times |V|$ but the vast majority of cells are 0 (very sparse). Notice again how similar words have similar vector patterns.

This is so because of two co-occurrence phenomena:

- Syntagmatic (first-order) association (surface similarity): sets of words all occur near each other, somewhat interchangeably. E.g. ‘wrote’, ‘book’, and ‘poem’ all tend to occur near each other so they are likely to have similar patterns (example: “Whether a book or a poem, what Jane Austen wrote will live for generations.”).
- Paradigmatic (second-order) association (paradigm similarity): words don’t necessarily occur near each other but nevertheless do have similar neighbors. E.g. ‘wrote’, ‘said’, ‘remarked’ all share a ‘paradigm’ of words they occur near. (example: “The candidate remarked that the troops were important.” “The candidate said he valued the importance of the troops.” “The candidate said the troops mattered a lot to him.”)

It has been observed that a narrow co-occurrence window (1-3) will tend to give words with similar *syntactic* properties more similar vectors and with a wider window (4-10) more

semantic and not necessarily syntactic similarity. Think ‘orange/apple/lemon/carrot’ for the former and ‘kill/death/killing/killed’ for the latter. These are not hard and fast rules.

Not all co-occurring words are equally informative! Consider ‘the’ and ‘of’ which occur many times very frequently with other words. It’s better to ask which words are particularly informative. Specifically, if words occur more frequently than they do by chance, this is interesting to us¹. We specifically define *pointwise mutual information* for words w_1, w_2 :

$$MI(w_1, w_2) = \log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)} \quad (17.1)$$

If w_1 and w_2 are IID, we’d expect $P(w_1, w_2) = P(w_1)P(w_2)$. If this is so, then $MI = 0$. If the words co-occur more likely than expected, i.e. $P(w_1, w_2) > P(w_1)P(w_2)$, then $MI > 0$. If they occur less frequently $MI < 0$. This last element is often ignored; we don’t really know what it means to be some degree of ‘unrelated’ plus the resolution needed to detect events *less* likely than the product of two events necessitates very large corpora. Typically we instead study *positive pointwise mutual information*:

$$PPMI(w_1, w_2) = \max(\log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)}, 0)$$

Here’s a worked example. Using the table above of frequency counts f_{ij} for word i in context of word j , we can calculate joint probability, word probability, and context probability, as:

$$\begin{aligned} p_{\text{joint}}(i, j) &= \frac{f_{ij}}{\sum_w \sum_c f_{wc}} \\ p_{\text{word}}(i) &= \frac{\sum_c f_{ic}}{\sum_w \sum_c f_{wc}} \\ p_{\text{context}}(j) &= \frac{\sum_w f_{wj}}{\sum_w \sum_c f_{wc}} \end{aligned}$$

	$p(w, c)$					$p(w)$
	computer	data	pinch	result	sugar	
apricot	0	0	.05	0	.05	.11
pineapple	0	0	.05	0	.05	.11
digital	.11	.05	0	.05	.05	.21
information	.05	.32	0	.21	0	.58
$p(c)$.16	.37	.11	.26	.11	

$$\text{PMI}(\text{information}, \text{data}) = \log \frac{.32}{.37 \times .58} = .57$$

Here are all PMIs:

	computer	data	pinch	result	sugar
apricot	0	0	2.25	0	2.25
pineapple	0	0	2.25	0	2.25
digital	1.66	-.56	0	-.07	0
information	-.8	.57	0	.47	0

¹see also TF*IDF, another way to formulate the same idea

To get PPMI, replace the negative values with 0. The unfilled boxes are also

17.2.6 Cosine similarity

A nice number for characterizing the closeness of two vectors is the *cosine* of these vectors. Each word is represented as a vector in $|V|$ -space. If the angle they make is small, the cosine is close to 1. Cosine is just a normalized dot-product. Simple dot product isn't a great way to calculate closeness, because longer vectors (i.e. with high values in some dimensions) will lead to larger dot product. Cosine normalizes this:

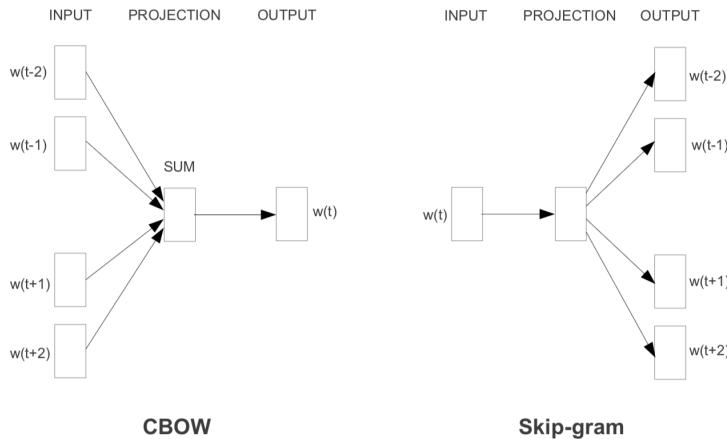
$$\cos(a, b) = \frac{a \cdot b}{|a||b|}$$

where

$$|x| = \sqrt{\sum_i x_i^2}$$

17.2.7 Neural(-inspired) distributional representations

An issue with PPMI is the vectors are very sparse and the dimensions very large. We previously saw embeddings were lower-dimensional dense representations of words. We want embeddings for words to be aware of the contexts in which these words occur. One way to do this is inspired by feed-forward neural networks. Mikolov's *skip-gram* is like a miniature version of the FFNNLM. It contains a word embedding matrix E and an output matrix O but no hidden matrix and no nonlinear function. Given a word w and its context word c , the logit for c is simply $(E_w O)_c$. Given some text, training data is formed by taking c to be any word within some range r before or after w . An alternative framework called the *continuous bag-of-words* sums together the embeddings of context words within r of w to predict it. In other words, for $r = 2$, the logit for w is $((E_{w-2} + E_{w-1} + E_{w+1} + E_{w+2})O)_w$.



These models, along with some techniques for training them very quickly, are known collectively as *Word2Vec* (w2v). Some nice properties observed with them is that one could do *vector math*; the vector formed by subtracting *big-biggest* is very similar to that formed from *small-smallest*. To this end, the W2v authors created an analogy test set. To evaluate

vectors, you consider an analogy like “brother:sister::grandson:granddaughter.” You calculate $grandson + (sister - brother)$. If the closest embedding to that vector is *granddaughter* the relationship has been captured. The relationship types are shown below, as are some results.

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Model Architecture	Semantic-Syntactic Word Relationship test set	
	Semantic Accuracy [%]	Syntactic Accuracy [%]
RNNLM	9	36
NNLM	23	53
CBOW	24	64
Skip-gram	55	59

It turns out the product of the w2v embeddings can be shown to be closely related to a PMI table.

Chapter 18

Information Extraction

18.1 Origins

(A lot of this comes from [10]. A lot also comes from Heng Ji's (UIUC) IE class slides.)

By 1994 the US government was already familiar with information retrieval: search a corpus of documents and retrieve those documents that match your search terms – now you don't have to read so many documents! But documents can be long; how about we search for just the facts we want instead? Originally there were templates of all the info the gov't wanted to find (e.g. locations and actions of ships scraped from navy telegraph cables). This was tested in a series of evaluations (this is how evaluations to drive NLP research got going!). Originally you had to participate in the entire pipeline of various kinds of information retrieval but then (1995) they split into independent and more generic tasks to encourage more participation by smaller teams. One of the tasks that year was 'named entity recognition.' This eventually led to the 'Automatic Content Extraction' program (ACE) which focused on even more fine-grained, independent tasks. The corpora produced in 2005 by ACE are still heavily used, nearly 15 years later.

18.2 Why

Why do we want to have a big knowledge graph at all? Ultimately, don't we just want to interact via natural language?

Yes, a really good version of e.g. Siri is an end goal. But even the super-awesome GPT-3 is not all that great at question answering. A fact repository, i.e. a knowledge graph, is fundamental to that kind of bot.

Consider also investigative journalism. In 2020, BuzzFeed got access to a lot of 'FinCEN' financial irregularity disclosures¹ which were a lot of financial transaction data but also narratives by bankers detailing the irregularities. The team actually tried to use IE but couldn't so they read everything by hand and compiled networks showing who was involved in what². It's not feasible to read in a database of reports and then write a news article;

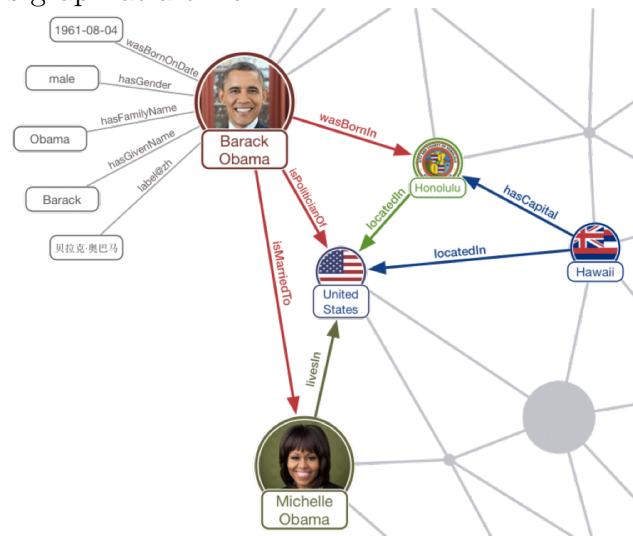
¹<https://www.buzzfeednews.com/fincen-files>

²<https://www.datanami.com/2020/09/25/icij-turns-to-big-data-tech-to-unravel-fincen-files/>

that's not likely in the current visible AI horizon. But it is feasible to be able to build an information network

18.3 Tasks

Here I'll mostly outline the individual tasks that we see today that are called IE and try to get into their details a bit. The figures show the types that were used in ACE and are still widely used today, though there are additions/exceptions. The ultimate output of IE can be thought of as a *knowledge graph*, that is, a representation of every conceivable piece of information and how it relates to every other conceivable piece of information, a snippet of which is shown below. With such a graph information queries would be more straight forward to look up. However in practice it is far more common to work on one aspect of constructing this graph at a time.



18.3.1 Named Entities

Type	Examples
person	Fred Smith; the undertaker
organization	Ford; San Francisco 49ers; a car manufacturer
GPE	France; Los Angeles
location	Nile; Mt. Everest; southern Africa
facility	Disneyland; the Berlin Wall; Aden's streets
vehicle	the U.S.S. Cole; the train; the helicopter
weapon	Anthrax; bullets; tear gas

The general idea here is to find all the things that have proper names. Names are not really part of the ‘core’ language so it makes sense to excerpt them. If we’ve POS-tagged then already we have an idea of what are names; they have NNP tags usually (not always). In general we’d also like all kinds of ‘language external’ elements like addresses, times of day, etc. and subsequent evaluations/definitions/systems have added more types.

But we don’t always just find the names. Sometimes we find the nominal references too (i.e. references to named things without using the names) as well as the pronominal references (references using a pronoun). Often these are evaluated separately; names are easier to detect and type than nominals, which are easier than pronominals.

In other circumstances we may want to find special purpose entities. One particular type of interest is chemicals, proteins, enzymes, etc. These are often identified in various kinds of medical literature.

Detection

The detection task is: given a text, find the spans of the entities. What constitutes a ‘span’ can be trickier to define than you might think. Consider:

The Los Angeles Times, a fine newspaper, arrives in my town on Saturdays, but it is usually late, because Culver City has a traffic problem and God hates me.

Some questions that have to be answered (typically consult your annotation guidelines; I provide my best guess):

- Is ‘Los Angeles Times’ marked? It’s the paper, not the organization, I think... (yes)
What about ‘newspaper’? Still an org? (yes?) What about ‘it’?
- Is the ‘The’ included? (yes)
- Is ‘Los Angeles’ the GPE marked? (No)
- Is the ‘City’ in ‘Culver City’ marked? (Yes)
- Do we count tokens or characters? (I think tokens)
- If tokens, is the comma marked? Or the period in ‘me.’? (no; there is a ‘standard’ tokenization)
- Is God a person? (No)

What makes matters worse is, despite annotation guidelines, people don’t necessarily read them, and might evaluate/tokenize differently, then report results that are not replicable. It can be a big mess! Assuming you can agree on those standards, Micro-F1 on exact span match is typically what is reported.

The best way to learn this data is as a tagging problem, but with tags like PER, ORG, GPE, LOC, FAC instead of NN, JJ, ADV, etc. And since the elements we are learning are multi-token, we use what is known as *BIO* notation; the beginning of an entity is tagged with B, and other tags of that entity are tagged with I. Words not in an entity are marked with O. Thus:

B-ORG I-ORG I-ORG I-ORG O B-GPE I-GPE O O O
The Los Angeles Times in El Segundo is great .

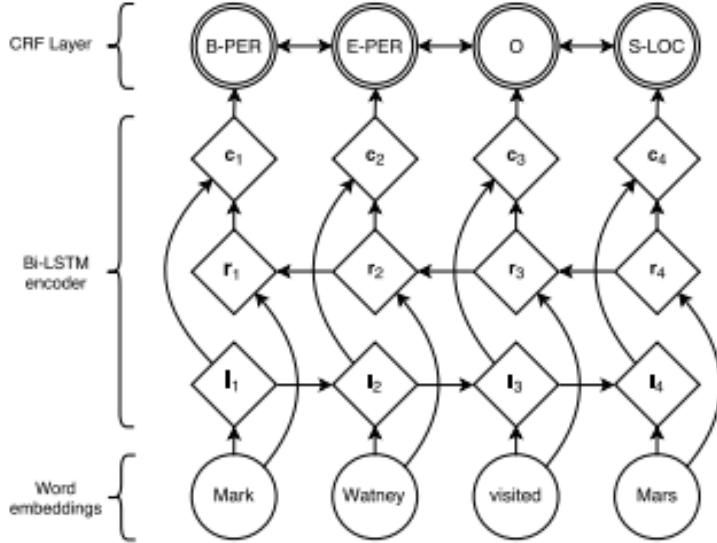
Another variant is *BIOSE* which distinguishes between Beginning, Inside, End, and Solo entities. Either can be learned with an HMM or LR/MaxEng tagger, just like a POS tagger. Even better is a CRF which considers the entire *sequence* to be an item that is predicted; for tag sequence $T = t_1, \dots, t_n$ and word sequence $W = w_1, \dots, w_n$:

$$P_{HMM}(T, W) = \prod_i P(w_i | t_i) P(t_i | t_{i-1})$$

$$P_{MEMM}(T|W) = \prod_i \frac{\exp(\theta \cdot f(t_i, t_{i-1}, w_i))}{\sum_t \exp(\theta \cdot f(t, t_{i-1}, w_i))}$$

$$P_{CRF}(T|W) = \frac{\prod_i \exp(\theta \cdot f(t_i, t_{i-1}, w_i))}{\sum_{T'} \prod_i \exp(\theta \cdot f(t'_i, t'_{i-1}, w_i))}$$

In the vanilla HMM we are just using word, tag and tag, tag co-occurrence, but in MEMM and CRF we can define various features. Naturally, those features could themselves be the outputs of a neural network, saving us the trouble of having to come up with features ourselves. A pretty good model [14] (90.94 on CoNLL 2003; SOTA as of 3/14/19 is 93.5 [3]) is the *bidirectional LSTM*; here's a figure from the paper:



They also use *character* embeddings; same idea. Note that these results are not hugely better than CRFs with hand-defined features, but no features needed to be defined. This works well across a variety of languages.

18.3.2 Coreference

Coreference is the task of clustering entity mentions that all refer to the same entity. This is particularly important when considering nominal and pronominal mentions but also when considering entities that might be named different ways (Los Angeles Times, The Los Angeles

Times, The Times, LA Times) or ambiguous people names (President Clinton, Secretary Clinton, Senator Clinton, Clinton). Here's an example from a recent survey paper [19] that illustrates how difficult this is:

The Queen Mother asked Queen Elizabeth II to transform her sister, Princess Margaret, into a viable princess by summoning a renowned speech therapist, Nancy Logue, to treat her speech impediment.

- How many people are described in the above sentence?
- Label all person mentions (name, nominal, pronoun) and indicate coreference

Features we are using as humans to get coreference right:

- gender match (Queen, her)
- number match (The team, The Yankees, they)
- entity type match (Joe Smith loves New York. The city this father of three grew up in...)
- metonymy – tricky! (The Yankees returned to their city winners. New York had lost the first two games...)

But the May 2 clash between separatists and Ukrainian government supporters in Odessa that took nearly 50 lives,... That battle was portrayed by Kremlin-controlled Russian media as evidence that the Kiev government is bent on recovering the occupied areas even if it has to shoot innocent bystanders to do so.

Is ‘government’ referring to the same government?

Pronouns in particular are not easily resolved with surface features; this has led to the ‘Winograd challenge’ exemplified as follows:

- The city council refused to give the demonstrators a permit because they feared violence.
- The city council refused to give the demonstrators a permit because they advocated violence.
- When Sue went to Nadia's home for dinner, she served sukiyaki au gratin.
- When Sue went to Nadia's home for dinner, she ate sukiyaki au gratin.
- Others, like agricultural entrepreneur Yevgeny Kremnyev of militant-occupied Svyatogorsk, voted for independence with the understanding that their autonomous republic will remain part of Ukraine but with more control over its foreign and economic policies. He said he intended to cast a ballot in the May 25 national presidential election, although he sees no candidate to his liking .

	PER			ORG			GPE		
	Chains not found	NEC F1	Coref F1	Chains not found	NEC F1	Coref F1	Chains not found	NEC F1	Coref F1
(Raghunathan et al., 2010)	16%	0.55	0.50	34%	0.42	0.41	14%	0.67	0.56
(Clark and Manning, 2015)	36%	0.46	0.56	40%	0.39	0.46	21%	0.61	0.61
(Clark and Manning, 2016a,b)	21%	0.61	0.67	29%	0.50	0.52	17%	0.68	0.65
(Lee et al., 2017)	28%	0.58	0.69	26%	0.58	0.56	12%	0.76	0.68
(Lee et al., 2018)	7.5%	0.80	0.77	15%	0.69	0.61	8%	0.81	0.69

Table 3: Performance of systems. Chains not found and NEC F1 refer to the new named entity focused metrics. Coref F1 refers to the evaluation combining MUC, B^3 and CEAFFE, on test data.

	PER			ORG			GPE		
	Name	Pronoun	Nominal	Name	Pronoun	Nominal	Name	Pronoun	Nominal
(Raghunathan et al., 2010)	0.55	0.45	0.23	0.47	0.35	0.11	0.73	0.44	0.19
(Clark and Manning, 2015)	0.50	0.34	0.10	0.46	0.34	0.15	0.65	0.57	0.22
(Clark and Manning, 2016a,b)	0.66	0.49	0.15	0.54	0.47	0.33	0.72	0.59	0.41
(Lee et al., 2017)	0.64	0.41	0.15	0.65	0.48	0.39	0.80	0.70	0.47
(Lee et al., 2018)	0.85	0.58	0.26	0.76	0.64	0.47	0.85	0.77	0.51

Table 4: NEC F1 by type of mention. The errors on names are high, though it is possible to resolve these with NER and string matching or similarity. Pronoun errors are high as expected.

Figure 18.1: Coref Results

I think even the best models are still not using all of the context available to humans (and not even all the context made available). It becomes a computationally explosive task and in a sense the search (attention?) through the information is not done in a principled way, yet.

18.3.3 Evaluation

While scores are calculated as F1, evaluating this properly is quite tricky since one has to consider how to deal with ‘polluted’ clusters, multiple clusters of the same entity, and absent clusters. Traditionally an average of 3 different F1 scores is taken.³

There are a variety of techniques for doing entity coreference. The top performer described in [1] [16] is built as a classifier on top of a bidirectional LSTM (fine tuned from ELMo); for each mention, a distribution over all possible antecedents is predicted. In general n^2 comparisons must be made!

Per [1] top Coreference scores on the Ontonotes data set (from 2007) are in the mid .70s to mid .80s for name mentions, depending on the type. .58 to .77 for nominals and .26 to .51 for pronominals. This is still very much an unsolved problem. It is also important to distinguish between coreference given perfect mentions and end-to-end coreference (i.e. after mention identification). See 18.1.

Grounding/Linking

Having formed a cluster of entity mentions, one may also link them to a pre-existing knowledge base and thus *ground* them to some already existing facts. A typical general purpose

³See [1] if interested.

knowledge base is Wikipedia; this is sometimes then known as *wikification*. Other than wikipedia, there are special-purpose KBs. One well-known one is ‘GeoNames’ which is a listing of 11 million placenames. Another is Chemanalyser, which has many chemical compounds, substances and compound classes. Freebase and YAGO are other all-purpose KBs.

Grounding is not at all trivial; consider *Chicago*, which can refer to a city, a band, a typeface, or one of many professional sports teams. In addition to grounding to the KB, there will generally be some entities that aren’t in the KB; these should be properly identified as NIL.

Note that it is possible to simply link without clustering first and if done properly all non-NIL links will also implicitly be proper clusters; this can often be easier than simply determining coreference. However, it is still necessary for NIL mentions.

As with coreference linking can be done with perfect mentions or end-to-end; SOTA is 86.6 micro end-to-end [13] and 94.9 with perfect mentions [26]; determining the methodologies is an exercise left to the reader.

Make It Harder

Some more challenging extensions on the basic entity identification task:

- Few-Shot: given 20 examples of a new type, how well can you recognize it?
- Cross-Lingual (often termed ‘zero-shot’): given training data in one language (typically English), how well can you do in another language where no training data is given?
- Cross-Domain: Turns out the way entities are mentioned and co-referred varies across domains (e.g. news text, conversation, web text)
- Fine-Grained: instead of 6–10 types, how about hundreds of types, e.g. `basketball-player`. Then the same span can have many different labels (think of them as properties)
- ultra-fine grained: extract thousands from large ontology (e.g. YAGO): predict location a mention should be in an ontology graph

18.3.4 Entity-Entity Relations (Relation Extraction)

Type	Examples
physical	location of a person: Fred was in France
part-whole	the lobby of the hotel; Paris, France
personal-social	his lawyer; his wife; his neighbor
org-affiliation	the CEO of Microsoft; a student at Harvard
agent-artifact	my home; my car
gen-affiliation	a Methodist minister; American troops

Relations are ways entities are related. Typically relations are between exactly two entities and the order matters. The table above lists relations from ACE (there are subtypes not listed); a SemEval task in 2010 [11] defines other relational types. There are special-purpose relation types too, e.g. in medical literature we often want to know about relations like ‘bonds’ or ‘phosphorylates.’

Basic Methods

Early methods that can work reasonably well are so-called “Hearst Patterns”:

1. Come up with some basic patterns, such as [PERSON], born in [LOCATION] for the “Born in” relation (think of some for “agent-artifact”, “part-whole.”)
2. Extract a lot of entity pairs from a large corpus (e.g. “Einstein, born in Germany”)
3. Find these entity pairs elsewhere in the text (e.g. “Einstein, who began life in Germany”)
4. Learn new patterns from these spans ([PERSON], who began life in [LOCATION])
5. Repeat!

Usually we want training data to do some form of supervised learning. The above method is a way of obtaining training data ‘automatically.’ Another general method is “Distant Supervision.”

1. Obtain a partial list of relations (e.g. from wikipedia)
2. Find sentences that contain the entities in the relation pair and label them.
3. (self-training extension) Build a model with the data, use it to label more data, build another model (this can get noisy and degrade quickly, though).

Prior NLP analysis, such as Entity mentions, coref, dependency parses, POS tags, and semantic relatedness are all good features that can be used to model this analysis task.

Outside Information

A major difficulty with relation identification (and IE in general) is that humans intuit information based on a lot of background or even common-sense knowledge, which is hard to convey to machines. Having access to *background information* makes the task a lot easier for machines. Here is a (made up) example:

David Cone was seen at the premiere of the new Star Trek movie last night. The former Royal and Yankee color commentator said science fiction was one of his biggest passions, along with his family and baseball.

There is a works-for relationship between David Cone and Kansas City Royals. There are many inference steps needed to make this connection. However, it’s relatively easy to find the wikipedia page for Cone, and then to find links for the Royals (and the Yankees) and this information makes it easier to elicit the relationship.

Some more examples of how it’s complicated to determine relations can be found in Figure 18.2.

Reasoning Types	%	Examples	
Pattern recognition	38.9	[1] <i>Me Musical Nephews</i> is a 1942 one-reel animated cartoon directed by Seymour Kneitel and animated by Tom Johnson and George Germanetti. [2] Jack Mercer and Jack Ward wrote the script; ... Relation: publication_date	Supporting Evidence: 1
Logical reasoning	26.6	[1] “Nisei” is the ninth episode of the third season of the American science fiction television series The X-Files. ... [3] It was directed by David Nutter, and written by Chris Carter, Frank Spotnitz and Howard Gordon. ... [8] The show centers on FBI special agents Fox Mulder (David Duchovny) and Dana Scully (Gillian Anderson) who work on cases linked to the paranormal, called X-Files. ... Relation: creator	Supporting Evidence: 1, 3, 8
Coreference reasoning	17.6	[1] <i>Dwight Tillary</i> is an American politician of the Democratic Party who is active in local politics of Cincinnati, Ohio. ... [3] He also holds a law degree from the University of Michigan Law School. [4] <i>Tillary</i> served as mayor of Cincinnati from 1991 to 1993. Relation: educated_at	Supporting Evidence: 1, 3
Common-sense reasoning	16.6	[1] <i>William Busac</i> (1020-1076), son of William I, Count of Eu, and his wife Lesceline. ... [4] <i>William</i> appealed to King Henry I of France, who gave him in marriage <i>Adelaide</i> , the heiress of the county of Soissons. [5] <i>Adelaide</i> was daughter of Renaud I, Count of Soissons, and Grand Master of the Hotel de France. ... [7] <i>William</i> and <i>Adelaide</i> had four children: ... Relation: spouse	Supporting Evidence: 4, 7

Figure 18.2: Example of complicated reasoning needed for learning relations

NN Methods

Grishman [10] reports that CNNs operating a fixed window covering gaps between entities are reasonable for relation detection. A graph convolution network over dependency structure may also do well (use the dependency links as adjacency instead of/in addition to the natural word adjacency.) BiLSTM classification has also been shown to work well. There are a variety of different kinds of test sets and top F1 performance seems to be in the mid-80s.

It can be really tough though! Figure 18.3 has some tough (for machine) examples.

Some reasons why it's tough

Pronoun referent

18.3.5 Extensions

- Cross-Document relation extraction (what are the set of relations expressed by considering a whole corpus, rather than a single document? Show supporting evidence)
- Cross-Lingual/Cross-Modality (Now the documents are in Chinese, Spanish, English, there are movies as well as text files...)
- Fine-grained relations (as with fine-grained entity types)
- Relation and Entity at the same time – maybe the decisions shouldn’t be pipelined but a joint decision should be made (can be done as semantic parse, can be done as multi-task model).

- People Magazine has confirmed that actress Julia Roberts has given birth to her third child a boy named **Henry Daniel Moder**. Henry was born Monday in Los Angeles and weighed 8 lbs. Roberts, 39, and husband **Danny Moder**, 38, are already parents to twins Hazel and Phinnaeus who were born in November...
- He [**Pascal Yoadimnadji**] has been evacuated to France on Wednesday after falling ill and slipping into a coma in Chad, Ambassador Moukhtar Wawa Dahab told The Associated Press. His wife, who accompanied Yoadimnadji to **Paris**, will repatriate his body to Chad, the amba. → is he dead? in Paris?
- Until last week, **Palin** was relatively unknown outside **Alaska**... → does she live in Alaska?
- The list says that the state is owed \$2,665,305 in personal income taxes by singer **Dionne Warwick** of South Orange, **N.J.**, with the tax lien dating back to 1997. → does she live in NJ?

Figure 18.3: Tough Entity Relations

18.3.6 Open IE

You need not have an ontology of relations, or even entity types for that matter, but could conceivably identify all relations of any kind between all entities in the world. An example from Stanford's webpage on open IE: the sentence “Barack Obama was born in Hawaii” would create a triple (**Barack Obama**; **was born in**; **Hawaii**),

In a survey on the matter [20] it is pointed out that a prime difficulty here is evaluation; data sets vary considerably and have very different properties, The goal of open IE is to scale to very large volumes of text but this makes preparing reference data difficult. Most evaluation sets remain in Wikipedia and News domains, even though another point was to be able to extract in heterogeneous domains. There is still a lot of research to be done

18.3.7 Events

Type	Examples
life	is born; marries; dies
movement	transport; travel
transaction	sell; purchase; acquire
business	found; merge
conflict	attack; demonstrate
contact	meet; phone; write
personell	hired; fired; elected
justice	arrest; trial; convict

Events are specific things that happen, involving participants, causing a change in state. There are various ways to define what constitutes an event, but in ACE there is typically a *trigger*, i.e. the word in a sentence that connotes the event, and zero or more *arguments*, that is, labeled spans of entities involved in the event. In an example from Eisenstein:

Elected mayor of Atlanta in 1973, Maynard Jackson was the first African American to serve as mayor of a major southern city.

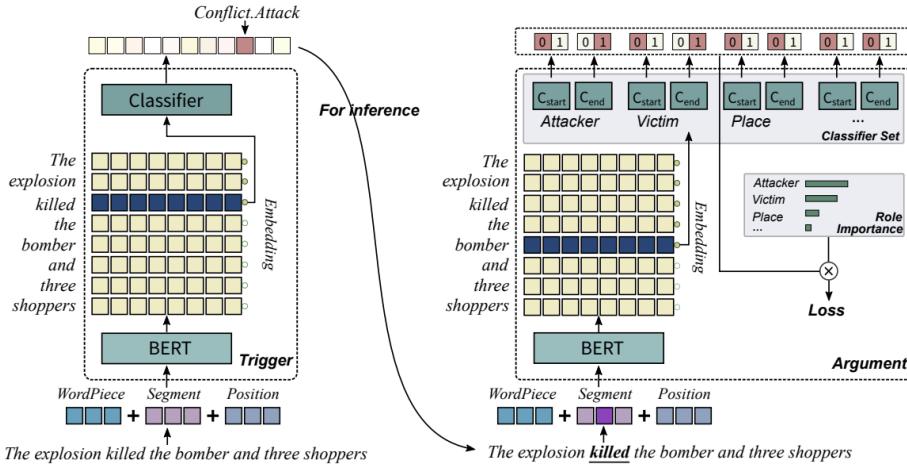
The event is an *election*, with roles office:mayor, district:atlanta, date:1973, and person-elected:Maynard Jackson.

Events in most data sets are discovered in the context of a single sentence. Multiple events can appear in one sentence, and the same span can serve as more than one argument in multiple events. For example:

John shot Mark for killing his brother Dan.

there is a *shot* event with agent:John and patient:Mark, as well as a *killing* event with agent:Mark and patient:Dan.

Various classifiers have been used to first detect and label triggers, then for each trigger detect and label spans. Trigger classification is in the 80s or so, and argument classification in the 50s-60s. Here is a figure and table of results from a paper at ACL in 2019:



Phase Model	Trigger Identification(%)			Trigger Classification(%)			Argument Identification(%)			Argument Classification(%)		
	P	R	F	P	R	F	P	R	F	P	R	F
Cross Event			N/A	68.7	68.9	68.8	50.9	49.7	50.3	45.1	44.1	44.6
Cross Entity			N/A	72.9	64.3	68.3	53.4	52.9	53.1	51.6	45.5	48.3
Max Entropy	76.9	65.0	70.4	73.7	62.3	67.5	69.8	47.9	56.8	64.7	44.4	52.7
DMCNN	80.4	67.7	73.5	75.6	63.6	69.1	68.8	51.9	59.1	62.2	46.9	53.5
JRNN	68.5	75.7	71.9	66.0	73.0	69.3	61.4	64.2	62.8	54.2	56.7	55.4
DMCNN-DS	79.7	69.6	74.3	75.7	66.0	70.5	71.4	56.9	63.3	62.8	50.1	55.7
ANN-FN			N/A	79.5	60.7	68.8				N/A		
ANN-AugATT			N/A	78.0	66.3	71.7				N/A		
PLMEE(-)	84.8	83.7	84.2	81.0	80.4	80.7	71.5	59.2	64.7	61.7	53.9	57.5
PLMEE							71.4	60.1	65.3	62.3	54.2	58.0

Table 2: Performance of all methods. Bold denotes the best result.

Events face a number of challenges. IAA is quite low on them in general. The largest data sets have fewer than 40k events annotated. Multilingual events fare even worse; ACE 2005 has Arabic and Chinese annotation as well but the amount is lower and the quality even worse.

How it's done

Methods are fairly similar to entity detection (neural CRf, lots of features, people trying BERT and its ilk). As with entities, syntactic and semantic features seem to help so pipelines are used which enable e.g. GCNs. Lexicons and external knowledge sources are also used.

Ontology

There are several. REO (rich event ontology) ACE (automatic content extraction) are popular. Could also consider the frames of FrameNet and verbs of VerbNet. The types are in the hundreds. Each event type has a set of valid arguments though most have an agent (often referred to as ‘arg0’) and patient (often referred to as ‘arg1’).

Evaluation

As with Entities, F-measure is used. However multiple steps are needed to ‘find’ an event and a portion of these could be considered ‘gold’, could be ignored, or could be included in the calculation:

- Find the trigger word or span
- Label the trigger
- Find the argument word or span (done multiple times)
- Label the argument (done for each found argument)

Often you will see argument and trigger statistics where the arguments are extracted given *gold* triggers. It’s important to read carefully!

18.3.8 Event-Event Relations

One can discuss relations between events as well. These are typically temporal or causal relations though there are also subpart relations and some others. Temporal seem to have been studied the most; the *Timebank* corpus [23] was constructed and annotated to try to capture temporally related events. An example is below:

```
There was <SIGNAL sid="8"> no </SIGNAL> <EVENT EID="57" CLASS="STATE" TENSE="PAST" ASPECT="NONE"> hint of trouble </EVENT> <SIGNAL id="11"> in </SIGNAL> the last <EVENT class="OCCURRENCE" aspect="NONE" eid="10" tense="NONE"> conversation </EVENT> between controllers and TWA pilot Steven Snyder. But <TIMEX3 TID="58" val="PT1M30S" type="DURATION" temporalFunction="false"> a minute and a half </TIMEX3> <SIGNAL SID="59"> later </SIGNAL>, a pilot from a nearby flight <EVENT aspect="NONE" eid="18" tense="PRESENT" CLASS="REPORTING"> calls </EVENT> in.

<SLINK eventInstanceId="eiid57" signalID="8" relType="NEGATIVE"/>
<TLINK eventInstanceId="eiid57" relatedToEvent="eiid10"
      signalID="s11" relType="IS_INCLUDED"/>
<TLINK eventInstanceId="eiid18" relatedToEvent="eiid10"
      signalID="s59" relType="AFTER" magnitude="t58"/>
```

How might events be temporally related?

- A started and finished before B
- A and B partially overlap
- A starts before and ends after B
- A and B completely coincide

However most of the time the start and/or end of an event is unspecified, unknown, or unknowable from text.

Events can also be structured relative to other events, e.g. an `election` is an event that involves, among other subevents, several `debates`.

Newer work [21] shows a variety of ways of looking at evaluation (there are again numerous data sets so timebank itself is not necessarily being used always); in general determining the order of events, whether they overlap or have containment relationships, etc. is not particularly agreed upon by annotators and thus it is difficult to build consensus in corpora let alone build satisfactory systems. Causality can be even worse.

18.3.9 Non-Events

A tricky bit about events is that while we use language to describe things that did or do happen, we also use it to describe events that are not so concrete:

- Events that did not happen ('The Dodgers failed to capture a World Series Title last night') – this example includes an event that didn't happen as well as a negative event (the failure) that did!
- Events that might happen ('If inflation rises too fast the economy could collapse, say experts.')
- Non-specific events that may have happened ('Whenever there is a fire, crew 147 is on the scene to put it out.')
- Hedged events ('These results suggest that the D gene might be involved in granulocyte differentiation')

In most event recognition work the task is to *avoid* recognizing these events and to only focus on actual events that occurred. To the degree that these not-quite-event cases are pursued, there is the FactBank (Sauri and Pustejovsky, 2009) which annotated 77,000 tokens and 9500 events on 208 documents for how factual the events are (factual, non-factual) and how certain the claim is (certain, probable, possible). They claim IAA of 0.81! I haven't seen a lot of recent work trying to solve this as a task.

18.3.10 Scripts/Schema Induction

A theory due to Roger Schank is the idea that all events comprise sequences of more fundamental events. These sequences are known as 'scripts.' The classic example is 'eating at a restaurant':

1. Enter = walk in, look at tables, figure out where to sit, sit.
2. Order = acquire menu, choose food, get waiter attention, provide order, waiter provides order to cook
3. Eat = cook gives food to waiter, waiter gives food to you, you put food in mouth, chew swallow

4. Exit = waiter gives check, you pay, waiter gives receipt, you stand up, you walk out.

A recent DARPA program is concerned with discovering these scripts (also known as schemas). It is quite difficult to find text evidence for these components despite having enormous corpora. Some of the more helpful data sets are instructional corpora such as WikiHow but even then there are many holes in knowledge.

There is prior empirical work in working with schemas. Chambers and Jurafsky extract *chains* of events from documents where the events have a common argument (a ‘main character’). From these chains then a Cloze test can be proposed, where one event is hidden and the goal is to determine what it is. This could be useful for narrative generation. There has been a little recent follow up work but the task still needs more definition and the quality of results indicate we’re still at the beginning of this complicated analysis.

Chapter 19

BERTology, or Blade Runner NLP

19.1 A Wide Variety of Semantic Tasks

- 1998: SENSEVAL: Word sense disambiguation shared task, coming out of the emergence of corpus-driven approaches. Good for semantics too!
- Blossomed into now annual multi-task workshop, eventually renamed SemEval – lots of word sense disambiguation tasks but then more and more generally ‘semantic’ tasks
- Shared tasks are a great way to establish baselines, gather data, drive interest in a new area
- I ran a couple of tasks in Abstract Meaning Representation parsing and generation, then co-chaired SemEval for two years; task approval is competitive, but then participation is not; good way to get into the NLP space!

Small sample of tasks

- WSD (Bass = fish, instrument, singer?). Variants in other languages, cross-language.
- Semantic Labeling (see connection to event recognition): Who is the agent and patient in “John shot Mark”?
- Detection and Interpretation of Puns (Is there a pun or not and where is it in “I used to be a banker but I lost interest”)
- Detection of Hate Speech/offensive language
- Some very special topics: “Extraction of Drug-Drug Interactions from Biomedical Texts” (“Cytadren accelerates the metabolism of dexamethasone”)
- Math Question Answering (like on the GRE)

19.2 Language Models are Good at Tasks

2018 (pub 2019): GLUE (see presentation) – suite of tasks. More or less designed for large language models such as ElMo, BERT, RoBERTa.

- COLA: Corpus of Linguistic Acceptability. Is this sentence good English (binary)
- SST-2: Stanford Sentiment Treebank. Positive or negative?
- MRPC: The Microsoft Research Paraphrase Corpus. Do these two sentences have the same meaning (binary)?
- QQP: The Quora Question Pairs. Do these two questions have the same meaning (binary)?
- STS-B: The Semantic Textual Similarity Benchmark. How similar are these sentences (5 point scale)
- MNLI: The Multi-Genre Natural Language Inference. Does premise entail hypothesis? (3-way)
- QNLI: The Stanford Question Answering Dataset (reformulated). Does the given sentence answer the given question?
- RTE: The Recognizing Textual Entailment. Entailment, 2-way.
- WNLI: The Winograd Schema Challenge (reformulated). Does the Winograd-posed question match the resolution?

By 2020 overall and on four of the tasks, SOTA performance exceeded human levels.

2019 (pub 2020): SuperGLUE – supposed to be even harder.

- BoolQ: Boolean Questions: Short passage and yes/no question.
 - Barq's – Barq's is an American soft drink. Its brand of root beer is notable for having caffeine. Barq's, created by Edward Barq and bottled since the turn of the 20th century, is owned by the Barq family but bottled by the Coca-Cola Company. It was known as Barq's Famous Olde Tyme Root Beer until 2012
 - is barq's root beer a pepsi product? (No)
- CB: CommitmentBank: some statement in a passage; how much does the author believe the statement?
 - B: And yet, uh, I we-, I hope to see employer based, you know, helping out. You know, child, uh, care centers at the place of employment and things like that, that will help out. A: Uh-huh. B: What do you think, do you think we are, setting a trend?
 - Hyp: They are setting a trend. (Entailment: Unknown)

- COPA: Choice of Plausible Alternatives: determine the cause or effect of a premise from two choices
 - Premise: My body cast a shadow over the grass. Question: What's the CAUSE for this?
 - Alternative 1: The sun was rising. Alternative 2: The grass was cut.
- MultiRC: Multi-Sentence Reading Comprehension: given a passage and some answers, choose which are true (0+ may be true)
 - Susan wanted to have a birthday party. She called all of her friends. She has five friends. Her mom said that Susan can invite them all to the party. Her first friend could not go to the party because she was sick. Her second friend was going out of town. Her third friend was not so sure if her parents would let her. The fourth friend said maybe. The fifth friend could go to the party for sure. Susan was a little sad. On the day of the party, all five friends showed up. Each friend had a present for Susan. Susan was happy and sent each friend a thank you card the next week
 - Question: Did Susan's sick friend recover? Candidate answers: Yes, she recovered (T), No (F), Yes (T), No, she didn't recover (F), Yes, she was at Susan's party (T)
- ReCoRD: Reading Comprehension with Commonsense Reasoning Dataset: multiple-choice Cloze test with entity masked out
 - Puerto Rico on Sunday overwhelmingly voted for statehood. But Congress, the only body that can approve new states, will ultimately decide whether the status of the US commonwealth changes. Ninety-seven percent of the votes in the nonbinding referendum favored statehood, an increase over the results of a 2012 referendum, official results from the State Electoral Commission show. It was the fifth such vote on statehood. “Today, we the people of Puerto Rico are sending a strong and clear message to the US Congress ... and to the world ... claiming our equal rights as American citizens,” Puerto Rico Gov. Ricardo Rossello said in a news release.
 - For one, they can truthfully say, “Don’t blame me, I didn’t vote for them,” when discussing the <placeholder> presidency (US)
- RTE: Recognizing Textual Entailment; given 2 statements, does the first entail the second (a, thus b)? Same as in GLUE.
 - A: Dana Reeve, the widow of the actor Christopher Reeve, has died of lung cancer at age 44, according to the Christopher Reeve Foundation.
 - B: Christopher Reeve had an accident (No)
- WiC: Word-in-Context: Is this word used with the same sense in both sentences?
 - Context 1: Room and board. Context 2: He nailed boards across the windows.

- Sense Match? (No)
- WSC: Winograd Schema Challenge: which phrase does the pronoun refer to? (was in GLUE in another form)
 - : Mark told Pete many lies about himself, which Pete included in his book. He should have been more truthful
 - Is ‘he’ Pete? (No)

January 2020 (and latest as of October 2020): still not better than human overall, but T5 is better on BoolQ, MultiRC, ReCoRD.

19.3 Designing Tasks that are Tough for Language Models

We’re continually trying to find tasks that humans can do well but that NLP models (particularly these large language models) aren’t able to do well. We’ve already seen that the basic IE tasks and machine translation are still not at fully human quality, but yet we press on with more tasks. Why?

- There are other aspects of language and cognition we’re trying to test. Analogues to child psychology studies – at what age do children learn to create nouns? when do they master casing, conjugation, etc? Psychologists make careful tests in order to draw conclusions about human development that can’t be directly observed. Our NLP “babies” similarly could be exposed to such tests. However we know that under typical circumstances humans will grow to ‘master’ the various capabilities being tested. We’re not yet at that point with NLP, seemingly, so we’re trying to find out what can be learned.
- Making tests and data sets is fun and publishable, and so is doing better on those tests. We’re all test-takers at heart, and we love to hill climb.

19.3.1 Cautionary Tale 1: ROC Stories

Researchers at the University of Rochester released an interesting corpus and task (Mostafazadeh et al. 2016). They asked annotators to write simple five-line stories, each with beginning, action, and ending. They asked other annotators to, when given the first four lines of the story, write a ‘good’ and a ‘bad’ fifth line. Mechanical Turk was used, but many safeguards were put in place.

- Had to judge five stories for quality or not (did they read the instructions carefully)
- Returned careful criticisms to workers
- Gave bonuses to top workers

Example:

- Line 1: Gina misplaced her phone at her grandparents.
- Line 2: It wasn't in the living room.
- Line 3: She had been in the car before napping in the living room.
- Line 4: She grabbed her dad's keys and ran outside.
- Possible Ending 1 (new author): But she didn't want her phone anymore.
- Possible Ending 2 (new author): She found her phone in the car.

(I couldn't find data that contained both the original line 5 and new author possible endings; it could be this wasn't released. As released, training data is just the original stories and dev/test is just the four + possible endings and the sets don't seem to overlap.)

This is a pretty useful and interesting data set! You need to use lots of discourse and semantic analysis to figure out which of the endings is the right one. But the shape of the task is very simple—it's just binary classification! Also the data set is potentially useful for training story generation systems. Because the stories are so simple, a lot of the complexity is removed, leaving only the core plot and conflict issues. This might also be good for event or event chain modeling!

However, a paper in 2017 (Schwartz et al, "The Effect of Different Writing Tasks on Linguistic Style: A Case Study of the ROC Story Cloze Task") found that with simple non-neural linear classifier, the right vs. wrong could be predicted at 65%, original vs right at 69% and original vs wrong at 76%...*without* seeing the context! This implies consistent stylistic differences and thus the test is not testing what you might think it's testing. In competition the best system, which used contexts, identified right ending at 75.2% accuracy, while a system without contexts still got 72.4%. Most differences do not seem intuitive! New answers are shorter, wrong answers start with NNP, right answers contain 'and.' The conclusion was that a great deal of care needs to be taken to test what you want to test.

19.3.2 Cautionary Tale 2: SWAG

Swag [35] is a nicely designed common sense test corpus. It presents a scene description and four options for what can come next. Here are some examples:

On stage, a woman takes a seat at the piano. She

- a) sits on a bench as her sister plays with the doll.
 - b) smiles with someone as the music plays.
 - c) is in the crowd, watching the dancers.
 - d) nervously sets her fingers on the keys.**
-

A girl is going across a set of monkey bars. She

- a) jumps up across the monkey bars.
 - b) struggles onto the monkey bars to grab her head.
 - c) gets to the end and stands on a wooden plank.**
 - d) jumps up and does a back flip.
-

The woman is now blow drying the dog. The dog

- a) is placed in the kennel next to a woman's feet.**
 - b) washes her face with the shampoo.
 - c) walks into frame and walks towards the dog.
 - d) tried to cut her face, so she is trying to do something very close to her face.
-

Table 1: Examples from **SWAG**; the correct answer is **bolded**. Adversarial Filtering ensures that stylistic models find all options equally appealing.

The general method is to find scene descriptions and examples of events that can come next in captioned video and movie descriptions. A generative language model (think neural MT where the input is in the same language) provided additional answers. Humans then filtered these to avoid having a set of gibberish and to mark answers that happened to be correct. The data as published had 85% (solo) / 88% (team) human performance but the best model they could apply (ELMo) only scored 43.6%.

But then BERT came along. Although SWAG was published first, BERT’s arxiv paper was well known before SWAG’s publication. BERT had 86.3% on SWAG! Did BERT learn common sense?

SWAG was followed up by HellaSwag [34] (originally known as SWAG AF, for...adversarial filtering). They analyzed why BERT did so well. They found that BERT was able to determine whether a sentence was *generated*, rather than whether it was sensible, and the nature of the test design led to the correlation of this classification and performance. This was the same test but had human performance of 95.6% and model (BERT) performance of < 50%, plus worse on ELMo, GLoVe, FastText, etc. What changed?

- No movie descriptions
- Data from how-to documents (wikihow) added; these have longer contexts than other data sets and appeared to be tougher in general for models to answer correctly.
- Candidate tests that were correctly classified by BERT were thrown out (!)

The paper points out that the adversarial filtering hurts all models, not just BERT, but it does seem like the goal posts have been moved.

19.3.3 More Tasks

There have been a lot of commonsense tasks proposed recently:

- MC-TACO (“Going on a vacation” takes longer than “Going for a walk”: A Study of Temporal Commonsense Understanding – Ben Zhou, Daniel Khashabi, Qiang Ning and Dan Roth, EMNLP 2019). Temporal Common Sense, dealing with events and time. E.g. “Paragraph: Growing up on a farm near St. Paul, L. Mark Bailey didn’t dream of becoming a judge. Question: How many years did it take for Mark to become a judge? 63 Years, 7 weeks, 7 years, 7 seconds, 7 hours” (Correct answer is 7 years, but I would say 63 years, and also that that could have been phrased better).
- PIQA: Physical Interaction: Question Answering (“PIQA: Reasoning about Physical Commonsense in Natural Language” – Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, Yejin Choi, AAAI 2020) asks questions or sets up tasks that require understanding of the physical world. e.g. “Make an outdoor pillow” – either “Blow into a tin can and tie with rubber band” or “Blow into a trash bag and tie with rubber band”.
- VCR: Visual Commonsense Reasoning (“From Recognition to Cognition: Visual Commonsense Reasoning” – Rowan Zellers, Yonatan Bisk, Ali Farhadi, Yejin Choi, CVPR 2019). Given an image, identified entities (and bounding boxes/outlines), and a question (e.g. “Why is [person4] pointing at [person1]?”) choose the answer from four options. Seems to require deep understanding. Currently best system is at 81.6% while human performance at 91%.
- NumerSense: Bill Yuchen Li’s work. (In 2020, Yuchen will tell us about it)

19.3.4 Discussion questions

- What is necessary and sufficient for a model to be said to have acquired some human skill (semantic understanding, common sense, recognition of acceptable language)?
- Should models be ‘allowed’ to have access to more data than humans have?
- How do you know when a *human* has mastered some skill? Is it sufficient to test them? To read an essay they wrote? To talk with them for a set amount of time?

Chapter 20

Creative Generation

Chapter 21

Dialogue

Chapter 22

Power and Ethics

There are entire courses on ethics in AI and NLP. A list of them is here: https://aclweb.org/aclwiki/Ethics_in_NLP. So this lecture will necessarily be incomplete. I'm also highly influenced here by a talk Alvin Grissom II (Ursinus College) gave at WiNLP in Summer, 2019. Also see Fairness in ML tutorial <https://mrtz.org/nips17/#/>. And Tsvetkov/Black course http://demo.clab.cs.cmu.edu/ethical_nlp/.

However, **The Views In These Lecture Notes are Entirely My Own**

22.1 Ethics

This covers a lot of ground, but consider some ways to argue:

- Utilitarianism – do whatever provides greatest good for greatest number of people (where ‘good’ = knowledge/pleasure/health/aesthetics). This takes society into account but can lead to some pretty awful behaviors
- Egoism – everybody works in their own self-interest. though not everyone knows what helps best or actively pursues it.
- So if you could choose what people see in a FB feed (using NLP), do you give them what they say they want or what will lead to overall harmony in society?
- Deontological approaches – Consider certain actions themselves to be simply good or bad. E.g. Laws of Robotics.

22.2 Bias/Discrimination

Shouldn't we discriminate? That's classification, right? That's what we've been trying to learn.

Counter argument: discrimination is appropriate when it is domain specific, not general. When irrelevant or, more importantly, historically unjustified/systematically adverse results have been used for discriminating, we can say, deontologically, we should stop using the current approaches.

22.2.1 Uncharged example: question answering with the wrong signal

The work referenced is [9].

SQuAD	
Context	In 1899, John Jacob Astor IV invested \$100,000 for Tesla to further develop and produce a new lighting system. Instead, Tesla used the money to fund his Colorado Springs experiments.
Original	What did Tesla spend Astor's money on ?
Reduced	did
Confidence	0.78 → 0.91

Figure 1: SQuAD example from the validation set. Given the original *Context*, the model makes the same correct prediction (“Colorado Springs experiments”) on the *Reduced* question as the *Original*, with even higher confidence. For humans, the reduced question, “did”, is nonsensical.

Question	Confidence
What did Tesla spend Astor's money on ?	0.78
What did Tesla Astor's money on ?	0.74
What did Tesla Astor's on ?	0.76
What did Tesla Astor's ?	0.80
did Tesla Astor's ?	0.87
did Tesla Astor's	0.82
did Astor's	0.89
did	0.91

Hopefully you agree that the wrong info is being used to make the right choice. And furthermore that this could very well lead to the wrong info being used to make the wrong choice.

22.2.2 More charged: Race and gender bias in NLP

[5]: pretty much every kind of bias you can imagine was observed in glove embeddings. Typical European-American names associated with pleasant words; black American names associated with negative words. Typical names for woman associated with arts; those for men associated with science.

Why is this a problem? For one thing, having stereotype biases, particularly strongly weighted ones, in your models, can lead to your models predicting the wrong thing, even if evidence beyond the bias counters the biased output.

Example: winograd test with bias potential [37]:

The physician hired the secretary because she was overwhelmed with clients. Who is overwhelmed? If replaced by ‘he’ are the models better able to predict? (or reverse and use was highly recommended.) If the sentence is Jill, the physician, hired Henry, the secretary because she ... you don’t need the end result to resolve the pronoun. Will the models resolve correctly? Default models evaluating on the ‘cross-bias’ set

are on average 21.1 worse in F1. Data augmentation (swap stereotypical entities in training data) mitigates...in that dimension. Gender is not binary, though binary gender dominates data and discussion. And what about e.g. race – much more than binary and more balance in this regard.

The counter argument is ‘people are biased, we’re just reflecting the data.’ Maybe we should do something about this! Consider an article about a black man stabbed by a white supremacist and how it ran in the New York Post:

Caughman, **who has 11 prior arrests**, walked for about a block after the stabbing and staggered into the Midtown South Precinct, looking for help. He died hours later after being rushed to a nearby hospital. Police sources said the **career criminal was refusing to talk to police about the incident and acting combative before his death**.

It doesn’t seem like this is necessarily limited to ‘known offenders.’ [4] argues that you can’t really create something without some intentionality:

A former Apple employee...described his experience on a team that was developing speech recognition for Siri... As they worked on several English dialects, he asked his boss: “What about African American English?” To which his boss responded: “Well, Apple products are for the premium market.”

22.2.3 Unintentional effects

COMPAS – a system for predicting probability of criminal reoffending. It was trained on a balanced data set, and race was not an input feature. However, ZIP code was, ZIP is highly correlated to race in the US, because of historical housing discrimination policies. Race is also highly correlated to socioeconomic difficulty, for the same reasons.

Additionally, the data was set up to predict whether a person would **commit a serious crime**. How was this judged? By who is likely to be **convicted**. Conviction rates are also correlated strongly with race.

We can talk about algorithms to debias these results. But people have to want to use them. If you’re trying to get a new SOTA on a GLUE task, and being biased helps because the *test set is biased*, what is the right move?

22.3 Power, i.e. Energy

A recent paper [29] analyzed what we’re doing in order to make deep learning nlp models.

Consumption	CO ₂ e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155

Table 1: Estimated CO₂ emissions from training common NLP models, compared to familiar consumption.¹

The big problem is the experimentation it takes to get to the final models. You’re constantly building and rebuilding, and the energy costs/CO₂ put into the air are tremendous. Here are breakdowns per model:

Model	Hardware	Power (W)	Hours	kWh·PUE	CO ₂ e	Cloud compute cost
T2T _{base}	P100x8	1415.78	12	27	26	\$41–\$140
T2T _{big}	P100x8	1515.43	84	201	192	\$289–\$981
ELMo	P100x3	517.66	336	275	262	\$433–\$1472
BERT _{base}	V100x64	12,041.51	79	1507	1438	\$3751–\$12,571
BERT _{base}	TPUv2x16	—	96	—	—	\$2074–\$6912
NAS	P100x8	1515.43	274,120	656,347	626,155	\$942,973–\$3,201,722
NAS	TPUv2x1	—	32,623	—	—	\$44,055–\$146,848
GPT-2	TPUv3x32	—	168	—	—	\$12,902–\$43,008

Table 3: Estimated cost of training a model in terms of CO₂ emissions (lbs) and cloud compute cost (USD).⁷ Power and carbon footprint are omitted for TPUs due to lack of public information on power draw for this hardware.

Maybe the energy’s clean? Depends where you live:

Consumer	Renew.	Gas	Coal	Nuc.
China	22%	3%	65%	4%
Germany	40%	7%	38%	13%
United States	17%	35%	27%	19%
Amazon-AWS	17%	24%	30%	26%
Google	56%	14%	15%	10%
Microsoft	32%	23%	31%	10%

Table 2: Percent energy sourced from: Renewable (e.g. hydro, solar, wind), natural gas, coal and nuclear for the top 3 cloud compute providers (Cook et al., 2017), compared to the United States,⁴ China⁵ and Germany (Burger, 2019).

There is also the problem that only companies really have access/money to train the truly big models.

What is the recommendation?

- report training time and sensitivity to hyperparameters. give a better sense of true cost
- government funded academic cloud compute: Academic researchers need equitable access to computation resources.
- Researchers should prioritize computationally efficient hardware and algorithms. No NAS!

22.4 Power, i.e. Control

22.4.1 Who funds your research?

In a University?

Then probably the federal government of the country you're in, and often the military. E.g. in the US the structure breaks down like this for CS:

- Company funding: 50-100k for 1 year. That funds part or most of a phd student, no conferences. Hard to support a phd since it's unstable funds. Gift, not constrained to a project
- NSF: 150-175/year for 3 years. Phd student plus a month of time and some travel. Decent way to support students. Fairly academically free but mission of the NSF is considered. Also, very very competitive.
- DARPA/IARPA: Can be 1m/year or more for 4 years. Funds a lab. But Defense/Intelligence have a specific task they want you to solve while you do research and you're tested on it frequently.

Unlimited rights of reuse are generally given to the funding agencies (esp. DARPA/IARPA). So be careful what you develop!

- Under counter-intelligence programs in the 50s–70s, US government spied on, harassed, and assassinated black and leftist activists
- FBI currently targeting "black identity extremists"
- What would they do with advanced NLP?
- Consider treatment of MLK by FBI under Hoover

In a company?

What is the mission of your company? If it's public, the mission **only will ever be to increase shareholder value**. If it's not, even then the ultimate goal will be to continue to exist; there is a hybrid utilitarian/egoistic argument to justify this.

It's hard to avoid being results-driven and the evidence shows that's what continues to happen:

- face recognition false positives on white male faces way less than other combinations. Do we expect this to be any different if detecting social media text and predicting malfeasance?

22.4.2 How will your research be used to exert power over others?

- Predictive policing - starting in the 90s, data-driven approaches ('Compstat') were used to use police more efficiently. However, this became more and more trusted by senior administrators and police changed their behavior to force the system to constantly show crime decreasing and more activity, by making increasingly meaningless arrests and not reporting crime. Since system sowed crime going down and arrests going up, things looked good.
- EMNLP Paper [7]. Extends work on predictive sentencing. Tries to predict the length of a sentence given the facts of a case in natural language and the charges. The paper argues accurate prediction rates, but what is the value of this paper if not to replace judgements by humans? And what is the value of a judgement by a human if not to find unique corner cases? An ethical statement is provided at the end of the paper arguing the technology should be used for 'review' only but will this happen?

22.4.3 Codes of Ethics

From Hal Daume (2016).

IEEE:

1. to accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment;
2. to avoid real or perceived conflicts of interest whenever possible, and to disclose them to affected parties when they do exist;
3. to be honest and realistic in stating claims or estimates based on available data;
4. to reject bribery in all its forms;
5. to improve the understanding of technology; its appropriate application, and potential consequences;
6. to maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations;
7. to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others;

8. to treat fairly all persons and to not engage in acts of discrimination based on race, religion, gender, disability, age, national origin, sexual orientation, gender identity, or gender expression;
9. to avoid injuring others, their property, reputation, or employment by false or malicious action;
10. to assist colleagues and co-workers in their professional development and to support them in following this code of ethics.

From Hal:

Responsibility to the Public:

1. Make research available to general public
2. Be honest and realistic in stating claims; ensure empirical bases and limitations are communicated appropriately
3. Only accept work and make statements on topics which you believe have competence to do
4. Contribute to society and human well-being, and minimize negative consequences of computing systems
5. Make reasonable effort to prevent misinterpretation of results
6. Make decisions consistent with safety, health and welfare of public
7. Improve understanding of technology, its application and its potential consequences (positive and negative)

Responsibility in Research:

1. Protect the personal identification of research subjects, and abide by informed consent
2. Conduct research honestly, avoiding plagiarism and fabrication of results
3. Cite prior work as appropriate
4. Preserve original data and documentation, and make available
5. Follow through on promises made in grant proposals and acknowledge support of sponsors
6. Avoid real or perceived COIs, disclose when they exist; reject bribery
7. Honor property rights, including copyrights and patents
8. Seek, accept and offer honest criticism of technical work; correct errors; provide appropriate professional review

Responsibility to Students, Colleagues, and other Researchers:

1. Recognize and properly attribute contributions of students; promote student contributions to research
2. No discrimination based on gender identity, gender expression, disability, marital status, race/ethnicity, class, politics, religion, national origin, sexual orientation, age, etc.
3. Teach students ethical responsibilities
4. Avoid injuring others, their property, reputation or employment by false or malicious action
5. Respect the privacy of others and honor confidentiality
6. Honor contracts, agreements and assigned responsibilities

Compliance with the code:

1. Uphold and promote the principles of this code
2. Treat violations of this code as inconsistent with membership in this organization

Deontological elements specific for NLP/linguistics

Support language variability and diversity

Recognize and model language as it is used

Respect the rights of humans to keep private language private

Chapter 23

How to Write a Paper

Graham Neubig <http://www.phontron.com/slides/neubig15paperwriting.pdf> is the basis for this.

23.1 Major Points

- Your goals are to be read by your reader
- Think about your reader above all else. Put yourself in their shoes and make life easy for them
- You have several readers: reviewer, adopter, future referent (20 year horizon)
- Do good work! Publish your accomplishments! But don't carve them up into small inconsequential pieces
- Read a lot of papers to get a sense of what works. Trust your instincts. The papers you like, where you read through to the end and learn something new, are very likely to be good papers. Copy their style!

You have been thinking about your problem for a very long time. Your reader has probably not been thinking about it and if they have they are surely using different terminology from you and have probably come to a different perspective than you. So you need to hold their hand:

1. Introduce the topic you are working on
2. Outline the basic problem and make a strong case for why it is a problem (there must be some kind of problem, or else there would be no paper. It can be a flaw in the way things are done today, a lack of prior careful analysis, or even a lack of compiled works. But the case should be made for why the reader should keep reading, i.e. why your paper should exist)
3. It is nearly certain that you are not the first person to work on this and that you are borrowing ideas from others. Make sure to connect those dots. This is not necessarily the same thing as a laundry list related work section (that comes later).

4. If you're going to get technical, use variables, use terms of art, etc. you need to create a closed environment in which your variables and terms can live. Define variables and functions at first use. Mechanisms that may not be obvious to everyone should be referenced, and if you're going to use the mechanism heavily you may need to briefly explain the working parts that you will manipulate.
5. If you're doing experiments, ensure reproducibility. You can do this by providing your code and data, but that is not sufficient. Explain the details such that someone could recreate *the experiment* from scratch. Note this is not the same as reproducing *your code* from scratch. Low level details like which data structures you use, descriptions of the code flow, whether the data is in xml or json format, etc. should not be in your paper.
6. Show your results. Numerical results are likely to require a table and/or graph. Make sure your significant digits make sense; you probably don't want more than one digit after the decimal point. Consider the right way to present results. Graphs should have the dependent variable (the variable you control) on the x-axis. Graphs should be line graphs if there is a trend along the dimension of the x-axis that can be reasonably inferred to hold in the points you don't evaluate. But if the x axis is e.g. names of different languages, a line graph doesn't make sense; use a bar graph. The results you present should tell a story that is intuitive before the details are inspected. The details, if you present them, should enhance the story and give nuance. They should be readable. If you're trying to get the reader to compare two numbers or points, those two numbers or points should be as close together as possible (this can be difficult in 2-d graphics, so it is something of an art). Don't expect any kind of comparison to be made between separate graphs/tables. The captions should be sufficiently self-contained to understand the basics of the story without the main text. The main text should reference the results.

23.2 Understanding Related Work

There is nothing new under the sun. Your work is related to others' work, and it's important to understand how it relates to that work and to let others know this as well. Why?

- Understand the history, allow others to build on your knowledge. Recognize the contributions of others
- Ensure you're not simply repeating old work and claiming it's new (sometimes it's ok to repeat old work, but you should be aware you are doing so)
- (sigh) Vain people want to make sure you acknowledge their contributions.

But don't *just* read! You may read too much and never get any work done on your own. Or you may come to the conclusion that everything has been done before. This is rarely true.

How to read?

1. Find a relevant paper. You may already be aware of one (e.g. you're trying to improve upon someone else's work directly). Or try searching for relevant terms in Google scholar. Or ask your advisor (they know what search terms to use).
2. (Forward-looking) Find papers that have cited the relevant paper. Look for ones that seem relevant. Also look for ones with fairly high citation counts. Google scholar, arxiv, other resources have indexed this.
3. (Backward-looking) Find important papers this cited. Understand the context in which it's cited; it could be refuting, or it could be not actually relevant. Skim through abstract to understand if you want to pursue. This is what bibliography is for! But it's also indexed by scholar, etc.
4. Repeat the process with forward/backward links.
5. Read through the papers that are most relevant, but understand the abstracts of others (don't read the whole thing)

23.3 Getting a Paper ‘In’

- Do good work! What’s good?
- Clarity: Is it easy to understand?
- Novelty: Is it new?
- Meaningful Comparison: Does it compare well with previous work?
- Reliability: Are equations and experiments correct?
- Impact: Will it make a big difference in the field?
- Replicability: Could others replicate the experiments?
- Overall Evaluation: What did you think? (what matters in the end)

23.4 Incremental work and ‘least publishable unit’

It might seem like a good idea to carve up your work as small as possible to get the most papers you can. After all, hiring/tenure committees seem to count beans this way. But your reputation will suffer; it's easy to see who is doing this. You're better off with fewer papers that are more highly cited.

23.5 Role of a paper in the future

Why are we writing papers? To communicate scientific ideas to people we are not talking directly to (otherwise we wouldn't need to write; Fun fact – Plato thought that writing was bad for knowledge, made us dumber because we didn't have to keep everything in our brains. He thought it was better to communicate all knowledge face to face).

23.5.1 reviewer

23.5.2 right after publication

23.5.3 In 3-4 years (scope of your dissertation)

23.5.4 In ten years

The field may have changed considerably. Your work will likely not be directly used but if you're lucky its successors will. So you're presenting

23.6 The structure of a paper

23.6.1 Abstract

23.6.2 Introduction

23.6.3 Related Work

Important detail point: should it be section 2 or section $n - 1$?

Can you get away with not having RW?

23.6.4 Preliminaries

23.6.5 Experiments/meat of the work

23.6.6 Results/analysis

23.6.7 Conclusion

Should there be future work here?

Maybe.

23.7 Presenting Your Work

23.7.1 Poster

23.7.2 Talk/Slides

23.8 Is this a defunct model?

- Seems weird to write a paper some times; maybe you only need a readme and a github
- that's probably true for using something right now. but probably a good idea to know how to contextualize the model/dataset/whatever being presented.
-
-
-
-
-
-

Bibliography

- [1] Oshin Agarwal et al. “Evaluation of named entity coreference”. In: *Proceedings of the Second Workshop on Computational Models of Reference, Anaphora and Coreference*. Minneapolis, USA: Association for Computational Linguistics, June 2019, pp. 1–7. DOI: 10.18653/v1/W19-2801. URL: <https://www.aclweb.org/anthology/W19-2801>.
- [2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].
- [3] Alexei Baevski et al. “Cloze-driven Pretraining of Self-attention Networks”. In: *CoRR* abs/1903.07785 (2019). arXiv: 1903 . 07785. URL: <http://arxiv.org/abs/1903.07785>.
- [4] Ruha Benjamin. *Race after technology: Abolitionist tools for the new jim code*. John Wiley & Sons, 2019.
- [5] Aylin Caliskan, Joanna J Bryson, and Arvind Narayanan. “Semantics derived automatically from language corpora contain human-like biases”. In: *Science* 356.6334 (2017), pp. 183–186.
- [6] Ciprian Chelba et al. *One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling*. 2013. arXiv: 1312.3005 [cs.CL].
- [7] Huajie Chen et al. “Charge-Based Prison Term Prediction with Deep Gating Network”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 6363–6368. DOI: 10.18653/v1/D19-1667. URL: <https://www.aclweb.org/anthology/D19-1667>.
- [8] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: <https://www.aclweb.org/anthology/N19-1423>.
- [9] Shi Feng et al. “Pathologies of Neural Models Make Interpretations Difficult”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (2018). DOI: 10.18653/v1/d18-1407. URL: <http://dx.doi.org/10.18653/v1/d18-1407>.

- [10] Ralph Grishman. “Twenty-five years of information extraction”. In: *Natural Language Engineering* 25.6 (2019), pp. 677–692. DOI: 10.1017/S1351324919000512.
- [11] Iris Hendrickx et al. “SemEval-2010 Task 8: Multi-Way Classification of Semantic Relations between Pairs of Nominals”. In: *Proceedings of the 5th International Workshop on Semantic Evaluation*. Uppsala, Sweden: Association for Computational Linguistics, July 2010, pp. 33–38. URL: <https://www.aclweb.org/anthology/S10-1006>.
- [12] Rafal Jozefowicz et al. *Exploring the Limits of Language Modeling*. 2016. arXiv: 1602.02410 [cs.CL].
- [13] Nikolaos Kolitsas, Octavian-Eugen Ganea, and Thomas Hofmann. “End-to-End Neural Entity Linking”. In: *Proceedings of the 22nd Conference on Computational Natural Language Learning*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 519–529. DOI: 10.18653/v1/K18-1050. URL: <https://www.aclweb.org/anthology/K18-1050>.
- [14] Guillaume Lample et al. “Neural Architectures for Named Entity Recognition”. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, June 2016, pp. 260–270. DOI: 10.18653/v1/N16-1030. URL: <https://www.aclweb.org/anthology/N16-1030>.
- [15] Zhenzhong Lan et al. “ALBERT: A Lite BERT for Self-supervised Learning of Language Representations”. In: *CoRR* abs/1909.11942 (2019). arXiv: 1909.11942. URL: <http://arxiv.org/abs/1909.11942>.
- [16] Kenton Lee, Luheng He, and Luke Zettlemoyer. “Higher-Order Coreference Resolution with Coarse-to-Fine Inference”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 687–692. DOI: 10.18653/v1/N18-2108. URL: <https://www.aclweb.org/anthology/N18-2108>.
- [17] Mike Lewis et al. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. 2019. arXiv: 1910.13461 [cs.CL].
- [18] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].
- [19] Vincent Ng. “Machine Learning for Entity Coreference Resolution: A Retrospective Look at Two Decades of Research”. In: *AAAI*. 2017.
- [20] Christina Niklaus et al. “A Survey on Open Information Extraction”. In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, Aug. 2018, pp. 3866–3878. URL: <https://www.aclweb.org/anthology/C18-1326>.

- [21] Qiang Ning et al. “Improving Temporal Relation Extraction with a Globally Acquired Statistical Resource”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 841–851. DOI: 10.18653/v1/N18-1077. URL: <https://www.aclweb.org/anthology/N18-1077>.
- [22] Matthew Peters et al. “Deep Contextualized Word Representations”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. URL: <https://www.aclweb.org/anthology/N18-1202>.
- [23] James Pustejovsky et al. “The TimeBank corpus”. In: *Proceedings of Corpus Linguistics* (Jan. 2003).
- [24] Alec Radford. “Improving Language Understanding by Generative Pre-Training”. In: 2018.
- [25] Colin Raffel et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *CoRR* abs/1910.10683 (2019). arXiv: 1910.10683. URL: <http://arxiv.org/abs/1910.10683>.
- [26] Jonathan Raiman and Olivier Raiman. “DeepType: Multilingual Entity Linking by Neural Type System Evolution”. In: *CoRR* abs/1802.01021 (2018). arXiv: 1802.01021. URL: <http://arxiv.org/abs/1802.01021>.
- [27] Victor Sanh et al. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *CoRR* abs/1910.01108 (2019). arXiv: 1910.01108. URL: <http://arxiv.org/abs/1910.01108>.
- [28] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Neural Machine Translation of Rare Words with Subword Units”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725. DOI: 10.18653/v1/P16-1162. URL: <https://www.aclweb.org/anthology/P16-1162>.
- [29] Emma Strubell, Ananya Ganesh, and Andrew McCallum. “Energy and Policy Considerations for Deep Learning in NLP”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 3645–3650. DOI: 10.18653/v1/P19-1355. URL: <https://www.aclweb.org/anthology/P19-1355>.
- [30] Ashish Vaswani et al. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: <http://arxiv.org/abs/1706.03762>.
- [31] Alex Wang et al. “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding”. In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP* (2018). DOI: 10.18653/v1/w18-5446. URL: <http://dx.doi.org/10.18653/v1/w18-5446>.

- [32] Yonghui Wu et al. *Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. 2016. arXiv: 1609.08144 [cs.CL].
- [33] Rowan Zellers et al. *Defending Against Neural Fake News*. 2019. arXiv: 1905.12616 [cs.CL].
- [34] Rowan Zellers et al. “HellaSwag: Can a Machine Really Finish Your Sentence?” In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (2019). DOI: 10.18653/v1/p19-1472. URL: <http://dx.doi.org/10.18653/v1/p19-1472>.
- [35] Rowan Zellers et al. “SWAG: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (2018). DOI: 10.18653/v1/d18-1009. URL: <http://dx.doi.org/10.18653/v1/d18-1009>.
- [36] Zhengyan Zhang et al. “ERNIE: Enhanced Language Representation with Informative Entities”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (2019). DOI: 10.18653/v1/p19-1139. URL: <http://dx.doi.org/10.18653/v1/p19-1139>.
- [37] Jieyu Zhao et al. “Gender Bias in Coreference Resolution: Evaluation and Debiasing Methods”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)* (2018). DOI: 10.18653/v1/n18-2003. URL: <http://dx.doi.org/10.18653/v1/N18-2003>.