NaturalLanguageProcessing-Lecture13

Instructor (Christopher Manning):Okay, hi. I'm back to continue reverting back to statistical parsing. So last time Dan Jurafsky talked about semantic role labeling. I mean I think for many people the idea of semantic role labeling is directly answering the kind of who did what to whom question, which is sort of the essence of having a kind of rough relational meaning taken out of sentences. And that's something that I'm not sure if he mentioned is there actually has been work done in some of these carnal evaluations as to well, can you get a bad representation successfully without doing syntactic parsing. And the answer to that seems to be a fairly large no so that for the kind of shared task evaluations and semantic role labeling that were done at the [inaudible] for two successive years, but for one year they imposed this constraint that you were not allowed to work with full pauses, that you were allowed to use part of speech information, partial chunking of to get base noun phrases, and there is low level kind of information, but you could not use full syntactic structures. And the results from that were just a lot worse than the results that people could get for doing semantic role labeling using syntactic structures. Now I'm personally a fan of parsing, but so you know I'm not completely unbiased, but I would argue that there's pretty strong evidence from that kind of evaluation that building syntactic structures is just a very effective means of working out the macro structure of the sentence, and that fits with how parsing has normally been motivated as essentially a prerequisite that you have to do before you can do deep level meaning extraction from sentences.

And the nice thing is that now that there are these quite available effective reasonable speed statistical pauses widely available, I think that you're starting to see more and more results of people, both inside NLP and people outside NLP and just political science or whatever that want to try to get meaning relationships out of sentences successfully avoiding syntactic pauses and haven't worked that out. Okay, so reverting back to statistical parsing, what I want to do today is first of all talk about this idea of lexicalized pauses which has been one of the key ideas that have been pursued in terms of improving statistical parsing. And then I'm gonna go on and say a bit more about ways that you can go about effectively solving the search problem of parsing.

And then next Monday I'll do the final lecture of parsing, and then that one I'll probably still be finishing off the parsing search techniques, and then I'll talk some about the dependency parsing, and then just a little bit about discriminative methods in parsing and why they're useful. Okay, so I kind of just mentioned the idea of lexicalization before, but the idea was this, that if you just know that you had a verb phrase with a verb and a prepositional phrase. Yeah, that's kind of grammatically well formed to build that structure, but it tells you nothing at all about whether that's a reasonable structure to build that you have none of the lexicalization that you had in the purely linear bigram or trigram model which told you is this a reasonable sequence of words, the kind of thing people say. And so the idea was well, can't we just capture that pretty well in a PCFG model if we say that for any phrase it can be fairly well represented in its meaning by its hid words. So a noun is the head of a noun phrase. The verb is the head of the verb phrase in the sentence. And so we kind of percolate these heads up, and so we now have this enormous space of nonterminals, which include a lexical head. Well, then as soon as we do that, we then have a verb phrase warped rewriting as a prepositional phrase into, and you can say well, walked into, that's a perfectly reasonable pair. And we have a PP into with the child of a noun phrase store, into store. That sounds reasonable, whereas if the sentence had been something like Sue walked onto the store, you'd say onto the store. That's kind of a little bit weird there. We wanted to prefer that kind of structure being built, okay.

And in particular, in the first electronic parsing, I talked quite a bit of this notion of lexicalization and attachment to pendencies, the prepositional phrases. And we can see how some of that is being captured as soon as we lexicalized these parsers. So announce rates for January, we've now got this bilexical probability between rates and for – and we have rates for January seems kind of reasonable. And here we have announce rates in January, so then we have announce in. And so the argument is announce in sounds fairly likely whereas announce for sounds not impossible, but less common. Conversely late for sounds very natural, but rates in doesn't sound so good. So we're capturing some of the information that seemed important for doing prepositional phrase attachment, but brackets, but we're not actually capturing all of it. So what information are we not capturing that is probably useful? I think – so the thing that we're not capturing is what's the role of the head of the noun phrase inside of the prepositional phrase, which is January. So there's also a lot of information for doing prepositional phrase attachments, but also looking at rates January, whether that's plausible, versus announce January. So here we've got announce January.

I mean in this example they're both January, so it doesn't actually distinguish. But in the general case, you can also get quite a bit of information looking at this, the head noun of the noun phrase inside of the prepositional phrase. And indeed, this kind of piece structure is essentially the one case in English where it seems like to do a really good job, you'd want to have three kind of a joint probability between three lexical heads. So you had both late, you're deciding between announcing January versus rates in January, and you actually look at those lexical triples, whereas the kind of lexicalized PCFG model everyone uses actually has lexical doubles. So they're often called bilexical probabilities, so you can announce in and in January, but you don't get the triple. Yeah, but in nearly all other cases, having those bilexical probabilities seems to give you all the information you need. Okay, so this is the story of what people felt they'd learned in the 1990's from doing the work in that decade that made board covered statistical parsing seem successful. The basic story people told was that there was basic PCFG parsing with just the categories and the Penn Treebank, and that sort of worked, but it really didn't work very well in terms of getting good parsers. And so then what you did is you put lexicalization into your parsers and that they worked great.

And so this is Eugene Charniak at a 2000 workshop talk. "To do better, it is necessary to condition probabilities on the actual words of the sentence. This makes the probabilities much tighter." And here's a good example of that. I'll show you another example in a minute. So one thing you can have in English is verb phrases that take two objects, so that's something like gave the man a book or told Fred the answer. And so that's actually distinctive property of Germanic languages. If you go to romance languages like French, you can't get those two objects, but, you know, it happens. How often does it happen? No, not very often, but well, what you can then find is if you actually know what the verb is, then you can – you know, for some verbs, give is the kind of verb it happens with, and so there's a reasonable probability there. And for said, it basically doesn't happen. And so you get much tighter, more precise probabilities if you know what the verb is. And similarly, Michael Collins sort of makes the basic contrast between basic PCFG versus lexicalized PCFG. And so this gives this kind of a history of what happened with statistical parsing. There were basic PCFG's, and so is the label, precision label VCO F1 measure I mentioned before, even though he labeled it accuracy. Basic PCFG is around 73 percent, and then people start exploring how to build lexicalized PCFG's. So David Makamen, who is actually Stanford's CSPHD student's work, was really a pioneering work on building lexicalized parsers.

So it was really a very ugly model in the details of how it worked, and so it got to 84 percent. And then there was the work of Michael Collins and Eugene Charniak who are essentially the two key people doing lexicalized statistical parsing in the 1990's going to '85, '86, '86, '88, '89. So that the key jump appeared to be lexicalization, and then there was kind of gradual refinement at the models, and they got a little bit better. Now as I explained last Monday when I talked about some of the work on accurate on lexicalized parsing, I mean in retrospect based on that work and some more work that I'll talk about today, I think that picture isn't quite right because well, what we showed was that we could get up to 86.3 percent by using a cleverly refined PCFG that just puts in some general syntactic context, but puts in lexicalization based on hid words that are content words whatsoever. And if you have that in mind, well, then the gains from lexicalization start looking kind of small, or actually better than these first two lexicalized parsers, and these ones are gaining sort of somewhere between 1 and 3 percent. And I'll come back to later talking a bit more about that. And we'll see that although lexicalization is useful and if clearly something you need and there is circumstances, it hasn't yet been able to be turned into quite the magic ingredient that clearly work made it look like it was.

Right, so first of all, I'm going to present the parser that Charniak did in 1997, so Charniak's parser was essentially the simplest, most straightforward way in which you could build PCFG, which was a lexicalized PCFG, and have it not completely uncomputable because of the enormous non-terminal space you get with lexicalization, okay. And it's in the style of just a basic generative model, so this is the tree we want to generate, and we're gonna generate it top down, though the actual parsing he does, of course, is bottom up. Okay, so the way to think about this is to assume that we're at some point in the tree, so we started generating a sentence headed rows, which rewrites as a noun phrase and a verb phrase headed by rows. And we know that our left kid is going to be a noun phrase. And so to do the model for the Charniak parser, you sort of have a starting point here, and then we're gonna do the expansion down here, and then essentially just two key probabilistic distribution figures to make each choice. So the first choice is to say, "Okay, here we have this noun phrase." This noun phrase is gonna have to be headed by some word to be a lexicalized PCFG, so what we want to do is have a probability distribution of a front word it's headed by. And so that's the one that's here, see. So what’s the probability of a certain a headword, given the parent head word the category that we’re giving a head to and the parent category?

So the information we’re using is the parent. It’s going to be the pendant of rows. It’s a noun phrase, and its parent category is a sentence. So you’re using those three bits of information, and then you’re choosing the head word by probability distribution. And so you’re getting a probability of it being profits versus some other word. Okay, so then at that point, we now have a lexicalized category, and so the next question is to say, “Okay, well, we’ve now got this lexicalized category. How are you going to expand it?” And so that’s then here D. So D is saying, “What’s the probability of an expansion role giving ahead its category and the parent category?” So this time the information you’re using is the category’s a noun phrase headed by profits, which is under a sentence, so it’s making use of the saying make use of the parent category to give a bit more information. That’s no longer making use of that, one of the probabilities of expanding in different ways. And so here the expansion that’s been chosen is adjective plural noun.

And I’ve written here profits. The fact that profits is there is just the terministic because we said here the profits is the head of the noun phrase, and so then for any particular expansion, it’s deterministically given by rule as to what is going to be the head word of that expansion. So this expansion is J, J, N, and S with this word being the head of it, so therefore we know that profits must be the head of this category. And so the thing that we still need to give a headword to is this guy over here. And at that point, we’re precisely back to the picture over here of where we have something and we want to work out its head, and so let’s just start repeating over. Now in this case here, I’ve already gone down to part of speech tags, and so we generate a headword, and then we’re done because it doesn’t expand phrasily any further. But in the general case, it would. And so you can imagine that sort of expanding over here where there’s more phrase structure left. But in general, you’re just automating, first of all, choose a head for phrase, then work out how it expands. One of those categories will be the head, and we know its headword. For the rest of them, choose head words for them, and then work out how they expand, and the expansions work out the head. So you just keep on having those two distributions beat off against each other heading down, and that gives you the probability distribution of the tree.

Okay, and so this is a couple more charts of some data to give you some idea about how much information you can get where. So if you think about the lexicalization here, lexicalization comes in in two places. One place it comes in is here, which is what’s the probability of having a noun phrase headed by propers, given that the verb that heads the sentence rose. So those are the kind of probabilities that I’ve used initially to motivate things, for examples like the prepositional phrase attachment. And they’re referred to in the literature as bilexical probabilities because they involve two words, that you’re conditioning one word on another word. Now you’re also conditioning it on other category stuff, but people kind of a lot of the time say, “Well, there’s some other category stuff.” But think of this mainly as a bilexical probability basically because the space of words is so large that that’s kind of the key thing to think about, whereas having one more, one less thing that you're conditioning on over here that's just the category like noun phrase doesn't matter so much.

But then for this one here, when you're working out how to expand NP propers, you're not using rows, you don't have any facts about the heads down here yet, so there's only one lexical item that's being used on the conditioning side. So you're letting the expansion depend on what the headword is, and so those are then referred to as – yes, unilexical, I guess you'd say. Unilexical probabilities are only one word involved, and both of those are useful. So the key case where knowing the headword is very useful is this case of looking at the way verb phrases expand. So different verbs take distinctively different patterns of arguments, and there's just a lot of information in knowing what the verb is to knowing what patterns and arguments you'd expect it to take. And so here are some examples of this. So the four verbs that are being shown are come, take, think, want, and then these are various expansions of verb phrase and seeing how commonly they occur. So one possibility is that the verb phrase is just a verb, there's nothing else in the verb phrase. So that's a completely intransitive verb, so that's something like I came, right, but came is just the verb phrase. And while that's fairly common for come, it's then rarer for some of these other verbs.

I mean you could think, right, this shouldn't occur at all. I take, that sort of sounds a bit funny because normally take is thought of as a verb in which you have to take something. It's a transitive verb. You kind of scratch your head, you might think or maybe in chess or something you can say, "I take," and there's no object involved there. But really the place where these turn up is because these phrase structure trees aren't modeling the deeper kinds of dependencies that are going on in the sentence where things have been moved around. So if you have a sentence like, "What did you take," when you're in the context of asking about a picnic or something, that in the phrase structure the verb phrase will only have take left in it because the thought that's the object has been moved up front. Anyway, moving along, if you then consider other patterns, here's the transitive verb pattern verb noun phrase. Take is basically a transitive verb, and so you see that. Thirty-two percent of expansions are here, whereas come basically isn't a transitive verb. You might wonder where that one comes from. I think where that one comes from is that there's this weird structure in English where various nouns are used as bare temporal modifiers.

So if you have something like, "I came last week," but last week is just a noun phrase, you know, you can say, "I came on Tuesday," or, "Caterpillars come in the spring," and you get a prepositional phrase expressing time. But you can also just use these bare temporal noun phrases in English like last week. And so I presume that that's where that 1.1 percent is coming from. Think barely ever occurs. Want, like I want a new bicycle, you can get transitively, but you also get a lot of different things. And then you can go through the other patterns, and so some of them are quite stunning. So for think, the basic pattern for think is that you're putting an S var clause or switch the sentence with either an overt or non-overt compliment, you know. I think that is stupid or I just think that is stupid, right or I think it's stupid that either verb count as an S var compliment. And that's actually basically just right there three quarters of the time, what you get to think, whereas for another verb like I want? The want doesn't take an S var complement, at least it doesn't except in very archaic forms of Shakespeare and English. I want that you leave me now or something like that, that is doesn't occur. And so there's kind of basically close to three orders of magnitude difference in the different expansions.

Okay, but then you also get these bilexical probabilities, and here's an example of some bilexical probabilities. So the example I had before was prices rose, I think. This is showing that prices fell. If you're using the Penn Treebank Wall Street Journal, which is a slightly biases corpus in terms of mainly covering financial news, but if either of the word rose or fell, if what pops into your head first for fell would be something like, "The child fell," or something like that. That's not what you find in the Wall Street Journal. The things that rise and fall are prices and stocks and a few other words like that. And so they're actually just these incredibly strong lexical probabilities. So this one's showing that if the verb is fell and it's taking a noun plural subject fully one-seventh of the time, the headword of the noun phrase is prices. I mean that's actually stunningly strong. But then you can look at what's the chance of finding prices with less information. If you just know it's a noun phrase subject of a – there's a plural noun. There's a 5 percent chance it's prices. Wall Street Journal talks about prices a lot. If you just know that it's a noun phrase subject that's a plural noun, but you don't know the verb is past. It's 2.5 percent. If you don't know – if it's just a plural noun phrase, but you don't know where it is in the sentence, you're down to 1.3 percent. So what you see is you kind of get less precise estimates as you use less conditioning information.

Okay, so the way that Charniak then built his model because I mean the thing that should still have been in your head is well, it's nice to write this, but how do we actually estimate this because they're all the same problems of estimating sparse probability distributions because this is essentially kind of just like a bigram model made approximately 2,500 times faster by also conditioning on a couple of categories. And so the way Charniak did that was he used linear interpolation, which you should remember well. And so this is the probability estimate that was used for the bilexical probabilities. So on the top one it's everything conditioned on everything maximum likelihood estimate. I'll mention that one in a minute. Then for this one what you're doing is you're throwing away the parent headword so that you're just conditioning on the current category and the parent category, so it's no longer bilexical at all. And then finally you're also throwing away the parent category. And what you've got here is just the tagging probability, which is what is the probability of a category being headed by a particular word? What's the chance of a noun phrase being headed by prices or what's the chance of a plural noun being prices?

Okay, and the interpolation was done in a slightly clever way in which these weren't just constants, but they were done as functions based on how much evidence you had of different conditioning context, okay. The one extra case in here was because it seems an enormous gap going from conditioning on the parent headwords, not actually conditioning on any words at all, Charniak tried to intersperse in here this extra case where he made semantic classes out of the parent headword, so this was done by distributional clustering algorithm. And so you condition distributional class of the parent headword rather than the actual individual parent head word. It turned out that doing that, he got very little value out of, and in general, to this day, most people who build statistical parsers just haven't bothered to do that. It's an interesting research area still. On the one hand, it seems like sure, you should be able to get good value from these kind of distributional classes, but on the other hand, in practice it's often seemed very hard to get that kind of value. Okay, so here's an example then of how this smoothing works out in practice. So this is considering two cases, so this was one that we had on the slide before of profits rose, and then here's the word corporate being a modifier profits down in the bottom left corner of my slide, okay. And so what we see in this example is that in the Wall Street Journal, I guess was it – so I think it's 1989 Wall Street Journal was when what Penn Treebank was made out of. And I guess 1989 must have been a down year for stocks because I showed you on the previous slide that if profits fail, that was 15 percent of what you saw for fail. It turns out that in the training data that people conventionally use for statistical parsers, this environment, profits rose, a subject noun phrase, doesn't occur at all. So you've got zero occurrences of it.

So this most detailed estimate is zero and then you're falling back to these effectively pretty imprecise estimates that based on the class of rose, there's a .3 percent chance of it being profits. And just based on it being a noun phrase and a S, it's .00, you know, these very small numbers, and it's very improbable. Here shows the opposite case of when the model does actually have something that it's seen before as the bilexical probability. So in the Wall Street Journal, if the noun is profits, the modifier corporate is just really common. In particular, if you have this quadruple here of an adjective modifying a noun, and the noun phrase is headed by profits, basically a quarter of the time it's corporate profits. So you get this really strong bilexical probability where the chance of an adjective being corporate based on less conditioning information, you have a lot leaner. Okay, and so that's the kind of things that get smoothed together with these linear interpolations. And on the one hand, obviously the linear interpolation is useful. On the other hand, there's something kind of very scary about how badly estimated statistically any of these quantities are because it just sort of feels like it's really roll the dice randomness that if you have something that's being seen as a bilexical probability, you're ending up. If you just imagine linearly interpolating these numbers using any reasonable weights, you're gonna end up with a really high probability here, whereas if you're linearly interpolating these weights, no matter how you weight the four terms, clearly you're gonna come up with something that's about two or three orders of magnitude smaller, which somehow seems quite a scary level of sort of bounciness in what comes out of things.

And so that's actually a dominant issue when you think about statistical parsing, but on the one hand, I think one can rightly tell the story that statistical parsing has just been really, really successful in delivering something that can reliably find pretty good parsers for sentences, but on the other hand, underneath that rosy story there's this picture of building parsers based on this overly small resource which really just can't tell you the kind of information that you'd like to know in a lot of cases. So the biggest resource for parsing is the Penn Treebank. And so the core pin – the Wall Street Journal Penn Treebank is one million words of Wall Street Journal. Well, recently they also passed some spoken material, the switchboard corpus, that is also one million words of switchboard material. But in practice, it's kind of hard to just bind the two together because there's so many differences with spoken material. People say I'm and are and a lot of my other things that's not really useful to just whack them together if what you want to do is pass sort of standard written text. So you've got about one million words of data. And one million words has been a classic size of corpora, so that when the brown corpus was made in the 60s that I mentioned earlier, it was also one million words in size. And as I think I mentioned then, at the time in the 1960s they were talking about the phenomenally large size of ground corpus and how it'd be able to answer any question about the English language. But that just didn't prove to be the case. And you can kind of see that for the Penn Treebank, so here are just a couple of slides that talk about how little information you have for estimating things in the Penn Treebank. So I mean so you've hopefully seen in programming Assignment 3 the kind of categories that we have for Penn Treebank parsing, which is the same as the ones you use.

So we did the parsing this year over this biomedical corpus, which is smaller again. It's only about 100,000 words of parse data, but it's using the same category said. And so as well as the standard noun phrase verb phrase, etcetera, categories, the Penn Treebank distinguishes special categories for WH phrases, so the simplest of which are who and what, but can be complex phrases like, "What kind of a person would whatever, steal money or something like that?" All right. So what kind of a person is then a WHNP in the Penn Treebank because a WH phrase is a noun phrase. So if you look at the Penn Treebank, the number of constituents phrasal categories is about the same as the number of words. There are about a million of those as well. But then if you ask the question, well, how many WH adjective phrases are there in the Penn Treebank, it turns out there are only 66 of those, and that doesn't sound so good for estimating probabilities of phrasal expansion. And then if you look a bit more carefully and you say, "Well, what of those 66 WH adjective phrases?" It turns out that 60 of them are either how much or how many because they're just the commonist to WH adjective phrases. How many books did you buy? But actually there's an infinite space of WH adjective phrases because you can take an adjective that can take compliments, such as clever, original, incompetent, silly, stupid, any of those evaluative ones that are positive or negative, and you can say, "How clever," and then you can put a compliment here as to what you're asking about. So how good at math is he or something like that?

You're in this infinite space of possible WH adjective phrases, and yet for this entire space, which includes this pattern and lots of other patterns, so another pattern you can have is for words like sure. They can take compliments, how sure that he will come are you? So there are lots of different kinds of complex WH adjective phrases, and in the entire Penn Treebank excluding how much and how many, you've seen only six of them, which just isn't much to go on to be building a model either. Here's another example of that with the bilexical probabilities. The whole basis of lexicalized parsing is we'd like to use these bilexical probabilities. And the question well, can we use and get value from them? And that's something I'll talk about more a bit later, but here are just some simple count statistics. So let's pick the most central most evidenced topic in the Wall Street Journal, which is what's happening to stocks, you know. That's just the most common thing to find data about. What do you find? Well, here are some counts for the stocks being subject of things and what happens, so stocks plummeted occurs twice. I guess it was a down year for the Wall Street Journal. Stocks stabilized occurs once, stocks skyrocketed occurs no times, and stocks discussed occurs zero times. And so apart from the fact that two isn't a big count, the central problem is that what you'd like to do is be able to tell these apart, that you'd like to know that stocks skyrocketed is a perfectly normal thing to say that you'd expect to be good, whereas stocks discussed shouldn't be okay. That just shouldn't be a possible sentence.

Okay, and so that kind of suggests that you just can't get the data to do this very successfully out of the Penn Treebank because the data just isn't there, that the Penn Treebank is much, much too small. And an obvious avenue to go about fixing that is to say, "Well, there's a lot of other data out there, like there are billions of words of data that are available as unsupervised data, and how can we get value from some of that to be improving parsers?" And I think that's a problem that's now starting to see a little bit of success, but it's taken a long time to have it show some success. So those only work again by Charniak in '97 in which he attempted to do precisely that, to improve his bilexical probabilities by using a lot more data, and essentially got no value out of it. But more recently, I should point out, McCluskey, who's a student of Charniak's at Brown, debated a paper on self-training the Charniak parsers. So what you are doing was essentially in a straightforward manner take trained Charniak parser, go and parse a lot of million words of more text, and your hypothesis there is that although it'll mess up in various ways, a lot of the time they'll get the answer right anyway. And so therefore you'll be able to learn a lot more facts about bilexical probabilities, which will usually be right.

Now obviously there are dangers there because what you learn suffer from any biases that are already present in the parser, but they actually managed to get good results from self-training. And so effectively this was the first paper that showed successful results from augmenting the training data beyond the Penn Treebank to actually produce a parser that works significantly better. Until this paper, the only other results were at best inconclusive, and at worst, negative. And the standard has essentially been that although it seems dumb and obviously too sparse, that everyone just trained off the Penn Treebank because it didn't work any better to do anything else. Okay, so something that it might be interesting to wonder is well, how easy is it to do this Penn Treebank parsing once you've lexicalized grammars? And the answer is that this is slightly subtle. I think lots of people miss this the first time because it kind of looks like, well, we've still got these three categories, and it's just like before. Is the parsing problem any worse?

And the answer is yeah, the parsing problem is a lot worse, and it can be worse in two ways. One way is to say, "Okay, I'm just gonna parse and my space of non-terminals is then a pair of my old non-terminals and a word." And so then our complexity naively would be like it was before, G cubed times N cubed, but really G is now of the size of the old grammar multiplied by the number of words. And so in terms of the old G that we used to use, this would be equivalent to G cubed times N to the 6th because there'd be an extra factor of N hidden in each one of those. So that's really bad news if we're into the sixth parser. And if you want to be a little bit cleverer than that, you can essentially say, “Well, let me just use the A, B, and C, and let me remember where the two headwords are in five phrases,” because it seems obviously wrong to have gone into the sixth because really one side or the other here must be the head, and that head will then continue up to the higher phrase. And so if I just remember where those two heads are, then I’ve got another two factors of N there, and so then I have a running kind of G cubed times N to the fifth, okay. So that’s a start. I shaved off one in there. And if you just naively – if you sort of organize things with headwords and naively run a lexicalized PCFG parser, it’s true in the fifth. So here’s a bottom up, exhaustive, lexicalized parser. This is a simple numbers for length of a sentence, and here’s time on a log scale. And look, that means it’s linear, so it’s polynomial time. And if you fit that, you’ve got an exponent here of 5.2.

There’s an interesting little fact there that I’m not gonna get to in detail. A lot of the time people are used to ON bounds being kind of worst counts bounds that you never get to practice. It turns out that in general in parsing if you just empirically put your numbers and then work out the best-fit exponent, it turns out that the best-fit exponent is actually slightly higher than the ON bound. And you might wonder how that can be since ON bounds are meant to be sort of the worst case of what it can be. The reason it turns out because it really just turns out for low level systems issues, so but are your thing gets bigger, but you are less successfully able to keep things inside your various one, two, three locations on your chip and things like that and you’ve got more memory non-locality going on, and all that kind of system stuff that people try and ignore when they’re doing AI kind of stuff. But it’s a fairly reliable result that you repeatedly see. Okay, so that’s sort of the worst case. I mean if you look at the algorithms that Charniak and Collins used, I mean as algorithms they actually are ON to the fifth algorithms. They are nothing more than that, and so all of the work is then turned into how can we do various kinds of heuristic search to make this run rapidly in practice. And both of them did a lot of clever work in how to get their parsers to run efficiently in practice, both in terms of run time and amount of memory needed.

You can do better than ON to the fifth parsing. So this means this interesting thread of work sensibly by Jason Eisner and Georgio Sutter, who’ve explored ways in which you can exploit some constraints in the class of grammars. So really all of the work that’s being done in lexicalized parsing, you can actually reformulate in a clever way and turn it into an ON to the fourth problem. And if you put a fairly small constraint on how you go about doing things, you can actually get back to ON cubed. I’m not gonna present that now, but I will next Monday when doing the final lecture, present essentially the same idea in the context of dependency parsing, and I’ll show their algorithm at work there. Okay, so let me now for a few minutes give a kind of just a quick tour of what’s happened in lexicalized probabilistic parsing after that Charniak 1997 model. So the Charniak model was very simple and straightforward. It’s easy to explain, but it used grandparent modes for conditioning just like we’re doing for our PCFG’s for Assignment 3, but it didn’t do any of the other things that we encourage you guys doing, such as it didn’t use any of these ideas of machovization to break up rules. And as I discussed in lecture last week that if you generate each child one at a time with no conditioning, that’s bad because you don’t have the conditioning information you need, but if you generate all of – if you generate all of – if you generate an entire PCFG rule jointly, well, then that induces unnecessary sparseness. So surely you can improve things by doing some in between level of machovization. So both Charniak and Collins and more recent parsers have done that.

Okay, and so the Collins parser, in particular, is viewed as a mark off process that generates each rule so that when you have a parent, the parent has a mocked head for the lexicalization, and then their other dependants to the right and other dependants to the left, and the way that the generation is done in the Collins model. As you start from here, you first generate the first right dependant, then the second one, then the third one, and you keep on going in a language model like way until you generate a stop probability saying, “Stop generating dependants on this side.” And then you go over here and you say, “Should I stop?” If you say no, you then generate a first left dependant. “Should I stop?” If you say no, you generate a second left dependant, and you generate left dependants until you decide to stop. Okay, so this is the kind of picture of the Collins model. So you start with a parent, which has both its headword and the headword’s part of speech tag. This seems a subtle little detail, but it turns out in retrospect when Collins and Charniak were competing trying to outscore each other on parsers that one of the little differences that turned out Old Testament be very important was that you can get quite a lot of value by not only head lexicalizing your grammar in terms of carrying around a headword, but also carrying around that headwords tag. And that’s principle. That’s useful mainly because it gives you a useful extra level of back off that if you can’t actually get information particularly from the world, you at least use the word’s part of speech tag.

Okay, so then you know what the – well, you don't have to generate the head. You know what is tagged and its word is gonna you may have to generate its category if you though where. Of the head is still itself a nonterminal. And then after that you start generating the children one at the time. And so the central thing here is in Collins’ model, you generate them one at a time, but he has a model actually of some categorization, which is which kinds of children you’d expect this head category given with a tag and word to generate. So this is kind of like this idea of if it’s give versus think versus come, you expect different kinds of dependents to be being generated, okay. And then so firstly there’s a model of some categorization, and then secondly, there’s kind of sort of a general model of how much stuff you’ve generated so far. So Collins doesn’t machovize the grammar in kind of an obvious way like we talked about for the PCFG where you’re sort of saying, “Well, I’ll condition on the previously generated dependent or the previous two generated dependents.”

Collins does something rather more complex, but it’s kind of an interesting thing to try and do which sort of relates to linguistic ideas. So his conditioning is – so when you generate OI, you’re not conditioning on LI minus 1 directly at 4. You’re conditioning on two things. You’re conditioning on unsatisfied parts of the subcategorization frame. So the subcat frame would say maybe that this word likes to take a noun phrase and the prepositional phrase with certain probabilities. And if you’ve already seen the noun phrase, the unsatisfied of subcategorization frame would be a PP. And you're conditioning on a generalized distance measure, and this distance measure says some general remarks about what you've generated so far. So it records whether you've generated nothing or something so far.

It records whether you've generated something that's headed by a verb so far because that's sort of taken as an indicator that you've generated something kind of big and heavy. It records whether you've generated any punctuation so far because if you've seen things like commas, that has a distinctive role. Okay, and so then for doing it when you're generating these nonterminals, you're effectively again doing these two probabilistic distributions, one of which doesn't have the words, and then one of which does. So first of all, there's a probability model of generating the next dependant, which is represented as a category and a head tag, and then there's the PMW, which is filling in okay. What's the probability of that category and tag being a particular word, hear John. So the second one is essentially exactly the same as Charniak's second probability that you're filling in the lexical item that's going to be the head, whereas up here you're doing this more articulated generation of the different dependants rather than simply having a simple PCFG role. Okay, and Collins essentially used the same kind of ideas of combining different probability distributions that in his model he actually used a back off model rather than using linear interpolation. So if you're generating the headword for the bilexical probabilities that these were the back off levels that were combined, and this was all kind of complex to look at.

But here we are. We're conditioning the word and these are all the things that we could condition it on, so this is what's the category, what's the tag. These are these kind of distance features. Have you seen the conjunction? Have you seen punctuation? You know, what the parent category is, what the headword and tag of the parent category is, stuff about the subcategorization on the size, and there's one that I can't remember what it is. But you get the general idea. So then if you back off, what you do is – the first back off is the only thing you throw away is conditioning on the headword. So you're generating the dependant word with all the other information about the headword. And then if this doesn't work, Collins immediately throws everything else away and you're generating a word based on just knowing what the tag is. So what's the chance of corporate given that's an adjective.

Okay, so another little bit about more info on the Collins model. So Collins in '97 actually produced three models, which he named 1, 2, and 3, which the model I presented so far is I guess closest to Model 2. But Model 1 had less conditioning, and then he had these progressively more complex models. So the kind of things he put in, actually he put in some interesting bits of linguistics to try and improve things. So the distance measure, which favors close attachments sensitive to punctuation. He put in certain state splits himself. So he distinguished four NP's with postmodifiers versus base NP's that only have premodifiers like that which also we got value from from our lexicalized parser. There's the coordination feature. Sentences with gap subjects, so that's like infinitives like, "I want to leave now." The "to leave now" is missing a subject, so they're distinguished from other sentences. There's the model of subcategorization that I've already vaguely mentioned. And then finally in Model 3, he started trying to deal with that problem of the example I mentioned of what did John take? The take there looks like it's an intransitive verb with nothing else in the verb phrase. And so the idea here is, well, the fact that we're not accounting for things like those WH questions where something that is an argument of the verb is being taken outside of the verb phrase is really messing with our probabilities because we say that take can be used in a transitive verb phrase when really it can't. It's just that its object has been moved elsewhere. And so we implement an idea that's been used in linguistics and it's essentially the idea that you still see an HPSG if you're someone that's done LING 120 of having gaps or traces that are then threaded through the categories to keep track of displaced constituents. So that's kind of a nice linguistic idea to try and do – actually, it never produced any positive value in terms of how well the model worked. Just evaluate it as parsing model with this labeled – precision labeled recall.

Now you could reasonably think that if you actually want to do meaning recovery or even the kind of semantic role wavelength where those long distance dependencies are important to get right, that this stuff would come in useful. But if you're only wanting to get your precision labeled recall higher, you didn't succeed in that part working. But all the rest of this stuff that went into Model 2 was useful and shows – and I think it does start to show that some more careful linguistic modeling does give you some value. Okay, but something that's been being interested people and pursued quite a bit was, well, bilexical statistics of the time that we claim to be central to these models, are they actually that important and useful for parsing at all? So one of the first pieces of work that addressed that was a piece of work by Dan Gaudier, who was the same Dan Gaudier who came up in the semantic role labeling. And so Dan Gaudier kind of wrote his own attempt at replicating Collins' model. And he then said, "Well, suppose I just take away the bilexical probabilities." You know, that's an appealing thing to do statistically because if you look at the probabilities in this model that these kinds of models typically have a couple of million parameters and really that a huge percentage of a bilexical probabilities because they are the maximally sparse one.

So you can take away huge number of the parameters from the model by taking out the bilexical probabilities. Let me try that and see what happens. And so he did that and he removed those bilexical probabilities that I showed you from PMW. And it turned out that the parsing accuracy only went down by half a percent, very little. That result wasn't completely convincing because for whatever reason Dan Gaudier's attempt to replicate Collins' parser wasn't actually as good as Collins' parser, that it sort of performed about a percent and a half lower, so it was kind of a little bit unclear as to well, for the other percent and a half that Collins is getting somehow, is that coming from lexicalization or is it not? But other evidence continued to mount that suggested, well, these bilexical probabilities unfortunately don't seem to be doing much. So David Chiang and Dan Bikel came up with the following results. So they – Dan Bikel worked on very closely emulating the Collins parser in a reimplimentation and successfully did that to get the same kind of results that Collins got.

And they tried out an experiment of well, rather than using this idea of the head of a phrase of nouns being the head of noun phrases and verbs being the head of verb phrases, what if I just always choose the leftmost word as the head of the rewrite and don't make use of any linguistic information? And so it sort of seems like if you do that a lot of the time your bilexical probabilities kind of should be crazy because they'll just be between two weird words. And well, it turned out that you could do that, and again, it seemed to make pretty little difference. So I guess here you're losing a bit more than a percent, a bit more than a percent. So you're only losing a percent and a bit from choosing kind of crazy, weird head words. And so here's then more work from Dan Bikel when – so this was then some later work when he'd done this full replication of the Collins parsing model. And so here is the fore model, which is working very nicely like the Collins model. And well, you could take away the bilexical probabilities altogether again, and now it's looking again basically essentially like Dan Gaudier's results, that you're only losing about half a percent by taking out the bilexical probabilities. And so that makes it look like the bilexical probabilities are almost useless, and so that was something that intrigued Dan Bikel and he went on to work on quite a bit further.

And so one of the things that he looked at was, well, when I'm running my parser, it's all the time looking up probabilities, and so it's saying, "Okay, I'm considering putting together this noun phrase with this verb, this sentence headed by a verb phrase. Let's look up the probability of that using my bilexical probabilities." And so let me just instrument my parser and look at when I've asked those kind of questions as to how does it actually answer them. To what extent does it back off in that three level back off that I showed you before? And well, what he found was that, well, actually it's using – it's answering the question with a bilexical probability only 1.5 percent of the time it's using the rich everything but bilexical probabilities to answer about 11 percent of the time. And 87 percent of the time is actually just backing off and is using the estimate of the probability of a word given the tag, and none of the other conditioning is in there.

Okay, so that makes it look like the bilexical probabilities aren't really playing much of a role because because of data sparseness that most of the time you just don't have any counts for the bilexical probabilities and so you have to back off. But you could object to that result and say, "Well, if you turn on exhaustive dumping of what one of these parsers are doing, most of the time they're pursuing completely crazy, weird, horrible parsers that look nothing like a possible sentence of English, that that's just kind of what these parsers do if you explore their search space." You might have noticed that with your parsers, so maybe that's a stupid question to ask, plus yeah, most of the time it's trying to pursue some really crazy theory, and so yes, you have no bilexical information on that, but that's a good reason because it's trying to pursue some crazy theory that would never turn up. And so therefore what we really want to do is look at the parsers that were actually the correct paths because maybe what happens is that when it's pursuing a correct parse path that most of the time it can use bilexical information, and it's only when it's doing crazy theories that it can't. And so that turns out to be the case. So if you measure that, what's the percentage of the time that bilexical statistics were available on decisions that were made in the paths that was the paths chosen by the parser? It turns out that bilexical probabilities we used almost 30 percent of the time, so about one time in three.

So that makes it look like, okay, bilexical probabilities are actually used a lot in the correct parse paths. But that then leaves still a remaining question, which is well, okay, if they're used a lot in correct parse paths, why is it that if you turn them off altogether you're only losing about half a percent? And so Dan Bikel did some further data analysis on that, and it turns out that a lot of the time that the lexicalized probabilities just aren't actually very different to the unlexicalized probabilities. And so this is essentially recreating the kind of idea that on delay the accurate unlexicalized parsing investigation, that a lot of the time you can predict what's good or not based on categories. And knowing the lexical item isn't significantly altering those probability distributions, that's it's really only in the minority of cases, that's cases like certain of these prepositional phrase attachments where there actually is information that really does markedly affect what decisions that you should be choosing. And the factors at the moment that you kind of – at the moment you've got this intersection of there are only a limited number of spaces where that information is useful, and then there's only a limited percentage of those where your statistics are available to actually do a good job of that. And the intersection of those two things leave you with a small space where you help. And that's what gives you about half a percent.

And well, the first half of that, I think, is just a fact of nature, but the second half of it is the part about your very sparse data. And that part you can at least help improve. And that's where this question about how can you actually get better estimates of bilexical probabilities, I think is still a good research area. Okay, so then very quickly the rest of lexicalized statistical parsing up to the current day. So the kind of degenerative lexicalized PCFG that was kind of the mainstay as the one that worked best and a lot of people used for a number of years was a later version that Charniak did in 2000. And he published it under a title of a Maximum Entropy Inspired Statistical Parser, which was sort of really misleading and confusing because it might have made you think that it has something to do with maximum entropy. But you've gotta pay attention to the inspired word. I mean really it was still just a generative parser and the supposed inspiration was he used some kind of clever ideas about how to do the smoothing of the probabilities where you're looking at ratios, ratios and probabilities between something that has conditioning and doesn't have conditioning. And somehow he's thought of that as sort of inspired by having with the maximum entry models.

And then it picked up other details, so I've already mentioned the conditioning also on the tags, [inaudible] rules like Collins did, and that got up to about 90 percent label precision and label recall. Much more recently than that, another interesting bit of work by [inaudible] Petrov and Dan Klein was that they essentially continued along the line of the accurate unlexicalized parsing and saying, "Well, maybe we can get a long way by just having a limited number of categories represented in phrasal trees and now going the whole route of lexicalization." But suppose we kind of freed up and got clever about how we did those categories. So rather than hand producing the grammar by doing careful category splits, suppose instead that we say that our backbone is Penn Treebank categories, but we're gonna expand this space by learning latent categories. So each phrasal category is a combination of the original Penn Treebank category plus some latent variable, which we're going to learn from data. And so this was done essentially by

EM learning. In practice, they did some clever stuff, but you'd have to go and read the paper if you want to learn the details of. I'm not gonna do it here. And the essential part that's cleverly done is that they used this category split merge procedure so that you're attempting splits of categories, so you're then evaluating whether the split proved to be useful, and if not, then you're backing it back out again.

And at any rate, the way they did that proved to work extremely well, whereas there had actually been someone else a year or two earlier who had naively done the okay, let's put a weighted variable here and just do basic EM over latent variable. And that basically didn't work very well at all whereas the kind of clever splitting procedure gave a lot of value. Okay, and so in particular from the clever splitting procedure you're only splitting the categories that benefit from splitting. So this is showing you in vague grammar how many subcategories they product, the different Penn Treebank categories. And so for the ones that don't have much variation in internal structure, so here you have WH adjective phrase and WH add phrase and S var Q and conjunction phrase, but they're not split at all. They're just all single category or they're split into two subcategories whereas over on this side the categories like noun phrase verb phrase and prepositional phrase, they're splitting them into somewhere around 30 subcategories. They also split the parts of speech, and so this is showing you some of the ways in which the parts of speech are being split.

And so a crucial thing to note at this point is that this isn't any more an unlexicalized grammar in the sense of the original unlexicalized parsing work, but the idea is you didn't really want to put in specific information about words. Here you clearly have specific information about words. So here you essentially have word classes, so you have a word class of your family names. Here's a word – two word classes that are locations.

These are kind of first words of two word locations, and these ones are second words of two word locations. Here are months, etcetera, like this. So you effectively have this grammar that's done over word classes. The interesting thing is that whereas word classes didn't seem to work very week in that Charniak '97 work, word classes of these guys seem to work very well indeed. I'm not quite sure what the difference is from that. Anyway, here are some final parsing results. So here's accurate unlexicalized parser again. This was what I just mentioned again that if you just sort of simply said, "Okay, for each category, let's put in a latent variable and assume that you have 16 subclasses of that category." Well, it sort of works and you could say this is a little bit better and it was done automatically rather than hand split categories. But it kind of doesn't really work much better. Here's the Charniak 2000 model. The interesting thing is that – here is that the Petrov and Klein work entirely beats the Charniak 2000 model, and so again this is this result that lexicalization is getting some value, but the value you're getting still seems to be pretty limited actually because this does have this word class information, and that word class information is useful.

But it's clearly got a lot less of that than a real lexicalization because if you remember back, categories – oh, I didn't say the way their algorithm is done, the categories could be split in two at most 64 subcategories. That was the sort of limit of splitting, and the practiced most categories had at most around 30 subclasses, so they’re fairly crude. Although they’re semantic classes, they’re fairly crude semantic classes. They’re not very fine grained, and yet, this works extremely well. But Charniak hasn’t stopped working on parsing, so the most recent – this is kind of effectively at the moment, state of the art best parsing numbers. So this is then taking Charniak’s 2000 parser and then putting a discriminative reranker onto it. And I’ll say a little bit about that next time. And so that thing gets the numbers up to around 92 percent, and that’s sort of the state of the art for parsing, which is, in some sense, given the difficulty of the task, pretty good, I reckon. Many people believed in the last ;90s that sort of numbers So the numbers around here, 86 or 87, was where you could get to giving a million words of Treebank data and talking about data sparseness as I’ve been talking about. But I mean actually getting from here to there that you’re getting kind of fairly close to getting a 50 percent error reduction, and so at some point, it seems like you’re just not going to be able to get further without having more data available. But this piece of work here is again in the space of the only training data used is the one million words of Penn Treebank data, and there isn’t any other information outside of that.

Okay, so then I want to go on a say a little bit about how do people go about parsing with search algorithms. I’m almost out of time, and so I’m not really gonna get to do very much of that, but maybe I’ll just spend a couple of minutes and sort of say a bit about the general problem setting and then start the next time, mention a couple of ways of doing it. Okay, so in general when you’re parsing, you’re fundamentally what we have is a search problem, right? So we have a grammar, we have a sentence, and we want to search to find something. The things that we could want to find is that we just want to find the best parse for sentences, so that’s what you’re doing the assignment and is the most common task. But there are other tasks we could have. You could want to find a bunch of reasonably good parsers for a sentence. This is often useful if you want to do further processing. So, for example, some of the semantic role labeling work takes a number of good parsers and tries to generate semantic roles based on that, and precisely because that gives you greater robustness because you’re not putting all of your eggs in one parse basket. That helps a bit.

Another thing you might want to do is find the total probability of all parsers licensed by the grammar. So this is both what you want if you want to use a parser as a language model for something like speech recognition or any other task like machine translation. It also turns out to be that this is the problem that you need to solve if you’re gonna build discriminate parsers. And so in general there’s a space of ways that you can do things. So we’ve looked only at CKY for finding the best parse and that’s what you do for the assignment. You can extend that to all of these problems. So you can extend CKY stall algorithms to do [inaudible] parsing. There’s a very naïve way to do it, which you can probably think up in your heads, which is you think, “Okay, each cell in my chart, instead of storing the best thing there, let me just store the K best things, and then just run the algorithm combining all of these K best things.” And you can do that, and that works correctly. But it has a high space and time cost because if you do that, you’re then combining K things with K things in each cell, and so you’ve got an extra factor of K squared. And that means that if you’d like to get – and 50 best parsers is a number people sometimes use, that means your parser is 2,500 times slower than it was when you were just getting one best, which is kind of unpleasant. It turns out that you can work out a much clever algorithm to find things than that, which and the central idea of that is you kind of do deltas, that you know that a second best parse is mainly gonna be like a best parse apart from different and one plates. And if you want to know about that, you have to go off and read this paper, but you can do clever ways of decoding K [inaudible] parsers.

And so you can then use similar kind of chart and CKY idea to work out total probability of all parsers, and that then gets referred to as the inside algorithm. And this is then effectively where you have the maximum CKY algorithm. You sum the probability of all analyses that you find, and you put that in the cell instead, and therefore you have the total probability of a cell. Okay, so CKY is a good thing to know about. But if you have an interest in finding just best parsers, which is what most of the time we're doing a lot of the time, there are a bunch of other ways that you can do it. You can use beam search again that I mentioned in the NT case, but you can also use ideas of a agenda based parsing, which is effectively ways of exploring different search strategies so that you're not doing a lot of work that's never gonna turn up in the final parse. And so next time I'll say a little bit about how agenda based parsing works. That's it for today except that Paul and Deirdre have the primary Assignment 2's to give back.

[End of Audio]

Duration: 75 minutes