NaturalLanguageProcessing-Lecture03

Instructor (Christopher Manning):Okay, hi, everyone. We'll get started again for CS224n. Okay, so what I want to do today is spend a bit of time finishing up the stuff about smoothing probability models that I didn't get to last time, and then after I've done that, you'll be kind of two-thirds qualified to do the first assignment.

The other thing I just want to remind people about is that we'll have section on Friday, and that section's a really good opportunity to kind of go through concretely and look through some of these smoothing models. And we have some nice Excel spreadsheets that Paul [inaudible] will show that goes through how some of these things work. And so that'll be useful for the assignment as well.

And then after that, I'm going to get into doing alignment models for machine translation, and so really the material today is the heart of what you have to do in the second assignment, so make sure you haven't completely forgotten it by the time the second assignment comes along.

And then this, then, is material that's covered in a couple of places that you should look at. One place you should look at is chapter 25 of Jurefsky and Martin, which is about machine translation. But another source that's really good to look at and is on the web page is that there's this machine translation workbook that was written by Kevin Knight.

So it was written as a kind of tutorial set of exercises for a machine translation workshop he ran a bunch of years ago, and it's sort of very concrete and works through examples, and it even has a bit of stuff on the language model side as well as the alignment model side. And so that's also a good thing to look through.

Okay, so where we were last time was this problem of smoothing. We've looked through some data, we have empirical counts of how often different outcomes occurred as to what words occurred after [inaudible] "the." And our idea is somehow we want to smooth, shave some probability mass off the things that – or count mass off the things that did occur, and so this is referred to often as discounting.

And then we kind of need to give it to the words that never appeared, because we have the idea that many of them are possible words that could appear; it's just our data is always too sparse.

And so then right at the end, I started talking about Good-Turing smoothing, and I'll just try and say this once more without completely stumbling over myself and see if I can get it right this time.

Yeah, I mean, it's somehow a little bit confusing, what you're counting, but it works out quite straightforwardly in practice. So we have a data set of C [inaudible] large C, it might be a million, 10 million tokens of words. So what we're gonna do is do C

experiments, where for each experiment we take one word in turn and hold it out. We call this a pseudo test set, so this is like a model of leave one out cross-validation.

And then we train a model – we consider a model trained on the remaining C minus one tokens and consider it predicting our one test token. And so then the questions that we ask ourselves is well, how often will our test token be one that we never saw in the training data?

Well, it'll be one that we never saw in the training data if it's a word that occurs only once in the training data because then when it's held out, it won't occur at all in the C minus one tokens, and then so it'll be a brand-new word. So the Good-Turing estimate for how often – the percentage of time and total, you're going to see some unknown word, is then the number of words that you saw once in the total training data divided through by the total number of words.

And that's sort of the event of any unseen token appears, not a particular unseen token appears, and I'll come back to that issue later.

Okay, well, what fraction of held-out words is seen K times in training? Well, a held-out word will have been seen K times [inaudible] the training data of this experiment, if in the entire corpus it occurred K plus one times, because then it would be the once that's in the held-out data and the K times in the reduced training data.

Well, how many such words are there? Well, there are NK plus one such words, so the total count mass of words – the total count mass of times in which the held-out data was seen K times in the training data is K plus one times NK plus one. And so then the fraction of the times that occurs is that divided by the total count of token C.

Okay, so if in the future we expect that percentage of the words to be those with training count K, [inaudible] there are actually then in K words with training count K, the way we come up with an estimate for them is to say the expected count of a word with training count K should be taking K plus one times NK plus one divided through by the number of words that occurred K times, which is then NK.

And so that bottom equation gives the discounted count to assigned words that occurred K times in the total training data. So for each – for words that occurred with some count in the training data, you're reestimating them based on words that occurred K plus one times in the total training data, following this leave one out thought experiment.

Does that sort of make sense? Okay. That sounds really good. If you just literally try and do that, you come up with a big problem. So if I – so I told you last time that the most common word in the Turing corpus was "the." And if I try it out and use this equation and say let's reestimate the discounted count and hence the probability of "the," what am I going to get out as the answer? Yeah?

Student: Zero?

Instructor (Christopher Manning):Zero. Okay, why do I do that? Well, it's because "the" is the most common word and so I say how many words occurred one time more often than "the?" The answer is zero. And so the enumerated is zero, and the answer is zero. So that's kind of crazy.

But you actually – so Good-Turing reweighting is based on a theorem. So this is the same Turing as in the famous Turing came up with this in early work. The actual method I'm not doing here that's the derivation of the Good-Turing theorem, it doesn't actually just have these empirical counts here. The actual theorem is in terms of model expectations. And so we're just – I'm just here sort of substituting in the actual empirical counts and doing this leave one out experiment.

And that works fine just to use the empirical counts for the words with low empirical counts, because there are lots of them and those estimates are reliable. But once you get out here, it works particularly crazily. I mean, it's most crazy for the most common word to be reestimated as zero, but in general the counts are just gonna bounce around once you get out to the tail.

So to actually make this usable, you have to do something different. And the most common method that's used, but not the only method, is this simple Good-Turing method, which was proposed by Gale and Sampson, and essentially what you're doing with that is that you are using – for low count words, you're using the empirical counts of counts, and for high count words, you're just fitting a little power law regression curve through the actual observed counts, and you're using that. And then since that's monotonically declining function, you're then always getting sensible discount estimates.

Okay, and so that's a little slide I'll leave – you can work through something in section.

Does this work well? It turns out it works fantastically well. So well, we discussed last time that another way that you could discount is to use held out data, so here what we've done is say well, let's initially do counts of bigrams on 22 million words, and then we'll look at a further 22 million words of held out data and see how often those bigrams actually occurred in further data. And these are the actual empirical counts.

On this side, we instead say no, let's just look at the empirical counts and then do simple Good-Turing smoothing. And what you can see is that the simple Good-Turing smoothing just does a fantastic job at matching what the empirical observed counts are.

So it's a good smoothing method, it's extremely widely used. I mean, and of course you could say well, if this way works so well, why don't we use this way of doing things? And the reason we prefer not to use held out data is in this experiment we've used – if we had 44 million words of data available to us and we used only half of it for training the model and half of it for doing held out discounting estimation.

And, well, if we have 44 million words of data available to us as I'll demonstrate in a few slides' time, we'd actually much prefer to use all 44 million words of data to train our

model. And by taking advantage of the Good-Turing smoothing idea, we are able to do that.

Okay. So just very briefly, then, I mean, Katz smoothing takes us into an area of backoff, and I want to say a couple of slides about backoff.

So a first approximation to estimate words, Katz uses Good-Turing smoothing. He actually uses a variant of Good-Turing smoothing, which is the following. So for these words here, here I fit a power law, but another way of working out the probability of these words is to say, well, the empirical estimates for high count words are basically just about right. There's no need to discount them whatsoever, because they're actually well estimated.

The probability of "the" that's occurring 4,000 times, we can just use relative frequency, and that'll be almost right. If you do that, since somehow you effectively have to discount a teeny bit more mass off the low count words so you end up with something that's still a probability distribution, which may be a little bit problematic, as I'll show later.

Okay, but Katz's main contribution is then how to make use of this idea of backoff, so that if we are trying to do bigram estimates and we've seen a bigram, well, then we'll use its discounted estimate using Good-Turing smoothing. Well, what if we haven't seen a bigram, so we're wanting to predict the probability of spacious following large, and we never saw spacious following large?

Well, one thing we could do is just put a uniform estimate over all words that we didn’t see following large, but that seems to be under-using the information that we have available. It seems like what we should do is at least use the information about what words are common or uncommon.

I use the unigram count of words. So in the absence of other evidence, we'd predict that large red should have higher probability than large spacious, simply because red has higher probability than spacious in our training data.

And so in Katz smoothing, what you're doing is you're using the Good-Turing discounted count of the bigram, combining that with an estimate from a unigram count, which is then being re-normalized to get a probability distribution.

Okay. And so that's essentially this intuition here, that you're using Good-Turing to assign probability to the things that you know about, and then for the things that you don't know about, Good-Turing tells you an aggregate of how much probability mass to give them, but you're then using backing off to a lower order engram to divide that mass among different possible word candidates.

Now there's a catch here, and the catch here is this works fine for words that you saw somewhere but just not in a particular context, right? The assumption of what I just said

before was that you had seen the words "spacious" and "red," you just hadn’t seen them after "large." So you could use their unigram counts.

Well, what about words you never saw in the training data at all? They're generally referred to in the speech and language world as "OOV" items – out of vocabulary items. And we still need to do something about those, we haven’t solved that problem, and I'll come back to that in a minute.

Okay, but before I do that, I'll introduce one and a half last smoothing ideas, and so this is the idea of kinesthenized smoothing. And so kinesthenized smoothing is sort of basically the state of the art these days, and the observation of kinesthenized smoothing is the following.

Well, this idea of backing off to lower order engram models, which we saw both in Katz smoothing and also in linear interpolation, there's something fundamentally broken about that. Because what happens is if you have a sentence up there with an unexpected and you're considering the probability of seeing "delay" or "Francisco" next, and you never saw unexpected "delay" or unexpected "Francisco" in your training corpus, then you back off from a bigram or trigram to the unigram and you're simply asking yourself which of these two words is most common as a unigram count.

And suppose, you know, quite likely for an American newspaper, if that's your data source, "Francisco" will actually have a much higher unigram count than "delay" does. And so you'll predict that you're more likely to see "Francisco" than "delay." And that seems fundamentally crazy, because actually, well, it says here that "Francisco" always follows "San;" that's not quite true, because you can also have Francisco as a name.

But 95 percent of the time in the newspaper, "Francisco" follows "San," so although it has high count mass as a unigram, we should really, really not expect to see it in that context.

And so the answer that's proposed to that was this idea of kinesthenized smoothing, and so kinesthenized smoothing takes that problem head-on and says, well, let me try and estimate how likely a word is to occur in novel contexts. And so how could we do that? Well, what we could do is think about how often did we see a word as a novel continuation?

I actually changed this slide from the one that was on the handout before. This is now the way it is in Jurefsky and Martin, chapter four, which I think is slightly more intelligible.

So what we're going to ask is for a particular word "W," how often did we see it making up a novel continuation; i.e., there was some word minus one before it, and then word "W" appeared. So, well that's the number of bigrams, word "W" minus one "W", which have a count of greater than zero; i.e., we saw them sometime.

Doesn't matter how many times we saw it, we've just seen it. So it turns up in a range of different contexts. And then we're just dividing that through by a normalization factor of doing that for all different words "W." So we're sort of saying of all of the times when some word first appeared after the preceding context, what percentage of that time was it a particular "W?"

So that's measuring how often does it occur in what new places where it hasn't been seen before. Does that make sense? And so that's proven to be a really powerful idea that's worked extremely successfully.

To make just one little side remark at this point, in a lot of areas of probability, everyone, especially in computer science, everyone is very into Bayesian stuff and doing Bayesian probability models. A kind of funny thing about the state of the art of using probabilistic smoothing in NLP is that all the really good ideas like this have actually come from people scratching their heads and looking at the data and what happens in the estimation and why does it go wrong, and by the seat of their pants, coming up with some formula that seems to capture the right properties.

And then what happens after that is then three years later, someone writes a conference paper saying how this formula can be interpreted as a Bayesian prior and does a big derivation of that, and that's been done for both Good-Turing smoothing and also just recently for kinesthenized smoothing. And so there's a link on the syllabus for a paper by [inaudible] of how to interpret kinesthenized smoothing as a Bayesian prior.

But the funny thing is that none of the actual good ideas of how to come up with a better smoothing method actually seem to come from the theory. They actually seem to come from people staring at bits of data.

Okay. Okay, so that's kinesthenized smoothing. There's a way that you can do kinesthenized smoothing which is actually a little bit simpler, and actually doesn't work so badly. This is the same chart I showed before of the actual discount from held out data and the Good-Turing estimate. I mean, something that you will notice, at least for this data set, is except for count one words – I mean, a way to – well, no, I'll say [inaudible] afterwards.

Except for count one words, look at this. It looks like the discount is basically three-quarters every time. And if you go up the higher counts, it essentially still looks like the discount is about three-quarters each time. So maybe this Good-Turing smoothing is more trouble than it's worth, and you could just subtract three-quarters from the observed count and you'll be near enough to write.

And that's referred to as absolute discounting. And it turns out that works pretty much as well, and so that's a slightly – then that gives you a sort of a slightly simplified version of kinesthenized smoothing where you're still using this continuation idea, but you're just putting in a constant discount there rather than doing Good-Turing smoothing.

Yeah, so coming back to here, I mean, yeah, the sense you get here is a word that you've seen only once, I mean, it's a little bit of evidence that you've seen it, that it's sort of kind of almost a random thing, right? You might have seen that word, you might have seen a different word.

So what Good-Turing smoothing gives you is words that only occurred once are strongly discounted, so the discounted counted it as less than a half, right? So it gets greatly marked down, whereas any higher counts kind of seem to have this basically constant discount.

Okay. So Stanley Chin and Josh Goodman have done a ton of careful empirical work, trying out every different smoothing method under the sun. This is one of the charts that appear in their paper as to how different things work. There's sort of a baseline, which is doing this held out smoothing. Here's Katz smoothing. Down here is then kinesthenized smoothing, and low is good, all right? The lower you are, the better.

And then they worked out their own modification of kinesthenized smoothing, which I'm not going to talk about, which is a fraction better than kinesthenized smoothing, but basically – [inaudible] – basically kinesthenized smoothing with Good-Turing discounting embedded in it is sort of the state-of-the-art smoothing language models. Yes?

Student:[Inaudible]

Instructor (Christopher Manning):[Inaudible] entropy from baseline bits per token. I.e., if you're reducing the – so this is on a scale across entropy, not perplexity, but you're wanting to make the cross-entropy as low as possible using the same kind of entropy or perplexity measures as last time. And this is just kind of taking a diff on that, rather than looking at the absolute number.

Okay. So that's kind of the smoothing technology. The other half of the story is language models work really well if you've got a ton of data. That may not be rocket science, it's true, but you know, this is kind of in some sense the slightly sad part of modern NLP, right? That it turns out that for a lot of things, you can try and do really clever things on a small amount of data, or else you can do dumb things on a very large amount of data. And, well, normally during dumb things on a large amount of data will work just as well.

Now of course you can have the best of both words and you can do clever things on large amounts of data and that will be better again, but you know, if you think about it abstractly, you can try and do a clever job at smoothing and you can do good things with that, but it's still kind of having a poke in the dark.

If your other choice is to go and collect the 100 times more data and then just have a much better sense of which words occur after large with what probability, that that empirical data is just going to be better.

And so that's kind of what this chart shows, so this is again on a bits of cross-entropy scale, and so we're showing different amounts of training data, 100,000 words, a million words, 10 million words, and the total data set – I forget the size now, I think it's around 50 million words [inaudible] in there.

And so what we see is – so for most of recorded history, by and large people have used trigram models, because it's obvious that unigram models are terrible. Bigram models are kind of passable, and in particular applications, it's still often the case that people will use bigram models. Because the counterpoint to my claim of lots of data is good is lots of data is good only if you've got data that's appropriate to the domain.

If what you're wanting to do is build a speech recognition system where people are going to be ordering Domino's pizzas, it doesn't really do much good throwing in 100 million words of Wall Street Journal Newswire. That's not going to help your language model.

So a lot of the time in spoken applications, around 100,000 words of data might be as much or more than the amount of domain-appropriate data that you can get your hands on. And well then, you know, absolutely you should just use a bigram model. There doesn't seem much value in doing anything else.

For most larger scale things, even just for regular kind of speech recognition like the $39.99 speech dictation software you can buy in the bookstore, the standard is trigram models because you're getting a noticeable performance boost by going to trigrams. And until quite recently, basically, people didn't go further than trigrams, because as you can see, these kinds of curves were very flat or actually things could even sometimes get worse, certainly with this Katz smoothing, from over-training.

But once you start getting into hundreds of millions of words or billions of words of training data, you can then clearly get value from fourgrams and start to get value from fivegrams. So a sort of big, huge MT systems use fivegram language models quite commonly now.

But you should have a sense of the fact that this is getting to be a very difficult game to play, because the problem is every time you're increasing the order of your mark-off model, that you're blowing out the number of parameters that you have to estimate by another factor of your vocabulary size. So if you want to have a vocabulary of something like 100,000 words, well, you have 100,000 minus one unigram parameters.

You have ten to the ten bigram parameters, ten to the 15 trigram parameters, you know, you very quickly get up into those kind of number of atoms in the universe kind of numbers, and you can't possibly be estimating them all well.

Okay, nevertheless, one of the things that's helped people a lot for doing large scale engram estimation is a year or so ago that Google released a huge set of engrams, going up to fivegrams, done over – [inaudible] this, this is a trillion – over a trillion words of running text.

Because the amount of data in the potential vocabulary was so high, I mean, actually, it's not exhaustive counts of all engrams or different orders. So in that size of data, they literally actually just threw away any word that didn't occur at least 200 times in their total corpus. And then even for the kind of bigrams, trigrams, etc., they're being trimmed unless they occur a baseline number of times of 40 times.

But nevertheless, you get these kind of enormous tables. This is a little estimate of how commonly words were seen after serve as the. And precisely, this is the way in which big data helps you a lot, because if your choice is you can use Google engram data and you have to use a fourgram language model and predict words occurring after serve as the, it's kind of a no-brainer that that's just going to have to give you way, way better estimates of what's going to appear next than having built your own language model over 10 million words of data and be using a trigram model which is trying to predict the next word based on the preceding context of "as the," because "as the" just isn't a very predictive context as to what's gonna come next.

Okay, there's one more slide on other things people do which I'm gonna skip. The last thing I wanna talk about for language models is this problem of unknown words. So what do we do about words that we never saw in our training data whatsoever? I mean, what is our story for those?

Well, our story for those actually depends on our application. There are some applications in which we just work with a fixed vocabulary, and so unknown words, by fiat, just don't exist. So that's actually typical in speed applications, right? That a speech recognizer has a vocabulary it knows about, those are the words for which it knows how they're pronounced, right?

If it doesn't know how a word is pronounced, it's not going to recognize it anyway. It's just not in the vocabulary, it doesn't exist. So you're building your language model only over words that you know about, and so those are referred to as closed vocabulary tasks.

But in a lot of other places, you're in the world of open vocabulary tasks, that you want to be predicting the probability of getting unknown words that you didn't see at all in the training data. And so they're referred to as OOVs and they're commonly represented as the token UNK sometimes, with XML-style angle brackets around it.

Well, so the way that this is most commonly done is to effectively regard UNK as an extra pseudo word, which UNK really represents the equivalent class of all never-before-seen words, right? So UNK – so when you're setting up a probability model, you can define your own event space, and so your event space here is okay, there are particular words that I know about that are in my vocabulary, and each time you see a word you're either seeing one of those words or you're seeing something else.

And the entire something else is bundled as a single event in your event space, and so that's then referred to as UNK. And so you can estimate a probability for UNK. You can use the Good-Turing idea for how likely you are to see novel words. You can do it using

held out data, by looking through some held out data and seeing how often you see novel words. But you can come up with an estimate for UNK.

If you do that, the estimate for UNK is going to be high, because since it's actually this equivalent class of anything previously unseen, you should expect that to be a kind of a big number. That is, its probability will be significantly higher when estimated than the probability of words that you've only seen once.

Okay, in many cases, people don't actually have only one UNK, they end up with several UNKs. They have one UNK for some unknown number or some unknown date or various other kinds of symbols, as well as things that are more words. Specifically for the programming assignment, it's just fine to say okay, we only have one UNK, that's all there is, and let's just work with that and do things.

This might all seem a little bit unsatisfying, and in a way it is a bit unsatisfying. You might wonder of, well, can we actually come up with estimates for individual new words? And you can, and sometimes people do. So one way that people do that is using spelling models, so that the idea is well, let me predict – I can't assume that the space of English letters is fixed, so let me assume that I can then kind of spell out new words a letter at a time. And so if I make a little character-level language model of sequences of letters, that will give me a probability of seeing any new letter sequence.

And so I can build such a model and divide the total mass of UNKs between those different things, and that's sometimes done. Another – if you do that, your models are really, really low in terms of your probabilities for each individual unseen word, because you're giving some probability match to every letter sequence, effectively.

So another idea that's certainly being pursued is to say well, can't we actually really estimate how big peoples' vocabularies are and give an estimate to unknown words? And there's actually a well-known – oh, man, I'm blanking on his name. What's the name of the guy in the Stanford stats department who did bootstrap methods? Students:

[Inaudible]

Instructor (Christopher Manning):Brad Ephron. Yeah, one of Brad Ephron's earliest papers is actually titled How Many Words Did Shakespeare Know? And the entire goal of the paper was to try and based on the number of words that are in the vocabulary of extant Shakespeare texts was to try and predict what his total vocabulary size was.

But, I mean, to summarize the paper very briefly, the short answer is kind of you can't. Depending on what parametric model you presume underlies the vocabulary, you just get estimates all over the place for his vocabulary, and you really can't estimate it accurately.

So really, if you want to play that game, there's sort of no – I think there's no better science than just sort of almost picking a number out of your hat of saying, hm, maybe the person knows twice as many words as we've actually observed.

Okay. So that's the end of language models, apart from I'll just show you one little side thing before I get into MT. So does everyone know what the Turing test is? A less-known fact is there's actually been an attempt for a bunch of years now to actually implement the Turing test, which has been referred to as the Loebner Prize. And the Loebner Prize was sponsored by Hugh Loebner, who's this guy who really has nothing to do with AI.

He made his money actually doing shows on Broadway, but somehow he had heard of the Turing test and he decided that he'd set up the Loebner Prize to have people compete to do the Turing test. And if you can succeed in doing it, it's got a million-dollar prize, but I guess the problem is that no one's really close to succeeding and the yearly prize for having the best – the most human-like computer of the ones that enter is only $2,000, and $2,000 isn't enough money to produce a lot of serious work. So it tends to be a bit of a hobbyist's competition of doing the Loebner Prize.

But nevertheless, the Loebner Prize has taught us some things that Alan Turing did not think of, and one of the things that it's taught us is that, you know, the Turing test, as he set it up, he kind of forgot the HCI angle. The kind of HCI – human communication angle of what happens when people communicate.

And so the way it's set up in the Loebner Prize, there are a bunch of judges and they communication by typing, so it's, you know, kind of like doing it with instant messaging. And on the other side, there might be either a human being who's the confederate or a computer.

And so there'll be some number of computers and an equal number of confederates in the competition, and the judge's job is to rank the confederates and computers in order of humanness. And if the judge is doing a good job, all the humans are above all the computers, and if the computer's succeeding in passing the Turing testing, the computer ranks above some of the humans.

Now, yeah, so the interesting thing that this has shown, really, is sort of the kind of social side of communication side of things, rather than actually the artificial intelligence. So although no computer in aggregate over the judges has come close to beating a human in terms of humanness, there actually have been multiple instances in which for some particular judge that they have judged a computer as more human than one of the human beings.

And so I guess you can kind of think of that as either the judge failing the Turing test, or you can think of the human being who's the confederate failing the Turing test because they couldn’t convince the judge that they were human enough.

One of the most celebrated instances of this, actually, was that the confederate was an English prof and they got on to the topic of Shakespeare. And after the end of each session, there's kind of – they collect debrief from the judges and the judge remarked "No human being could possibly know as much about Shakespeare as that," and he was graded down beneath a couple of computers.

And yeah, and so other stuff has been discovered. So one of the people who's done quite a bit of – done some of these hobbyist entries is one of my fellow compatriots, Jason Hutchens, who did this for a number of years. So one year he won for being the most human computer, and that wasn't the entry that I'm showing here. For that entry, and I think that this is true for quite a number of the systems that have done best in this, effectively they kind of souped-up [inaudible] systems which do a lot of template matching and then give appropriate answers.

And essentially what's been discovered is that the way you can kind of do well in this competition is by manipulating the communication and dialogue. So to the extent that the computer can seize control of the conversation, you can kind of keep up a pretty good charade of appearing human. And so essentially you portray a sort of a slightly schizo, jumpy personality, and the human being says something and you say "Oh, I'm not interested in that; have you seen any good movies lately?"

And they say something, and you say "Huh, I'm not into movies like that. What about Full Metal Jacket?" And they say something and then you say "Well, we shouldn't spend all the time talking about movies, are you into sports?"

And to the extent that you can kind of keep that up for a few minutes, you can sound actually surprisingly human. And Jason did that one year and won, but at the time, as a bright, young, interested in these kind of things student, he tried to do different things each year. And another year, he came up with this system, which was called Mega-Hal. And here's Mega-Hal at work.

Who won the Nobel Prize in physics? Heike Kamerlingh Onnes won the Nobel Prize. What did Kamerlingh Onnes discover? Heike Kamerlingh Onnes discovered superconductivity in 1911. Who explained superconductivity? Werner Hanke explained superconductivity in 1911. Who is Scarpino? Werner Hanke explained superconductivity to Scarpino. Did Scarpino win the Nobel Prize? The Nobel Prize for Physics in 1900 was won by Rontgen. Various other staff, the theorem of Pythagoras. Pythagoras sacrificed 100 cows after discovering his theorem. [Inaudible.]

My Italian's not very good. So what do you think Mega-Hal is doing here? Can you give a, can you kind of suss out what it's doing? Yeah?

Student:[Inaudible]

Instructor (Christopher Manning):No. It may have had some of that data in it. So what it's doing is it's being an engram language model. It's a pair of engram language models which have a lot of data in it, some of which may well have been derived from [inaudible]. And so what it's doing is for each time, it's taking the user's sentence, it's finding one content word in the user's sentence, and then it's running a mark-off engram model in both directions, so there are two engram models.

One that can run forwards and one that run backwards in both directions, and they stop after a bit, and that gives the sentence, and that's the reply that it gives. And some of them are a bit crazy. This entry wasn't as successful as the more [inaudible] like one, but it's sort of put-together text. Yeah.

Student: Wouldn't you just look for a Wikipedia [inaudible] instead of doing the [inaudible]?

Instructor (Christopher Manning):I mean, in 2008 I think that would be a very successful strategy. I mean, this is actually from – to be honest, I think this is around '97, '98. You know, it's actually now a bunch of years ago. The longer story of Jason Hutchens was after he graduated as a – this is a salutary tale. After he graduated with his Ph.D., he went and worked at an AI game company, and he seems to have gone from there.

I guess I tried to look him up on the web when I was doing this lecture and it seemed like the only thing I could find that he was doing now was writing a cooking blog. So be warned, if you go and work for an AI game company, you'll end up writing cooking blogs. Yeah?

Student: What is the [inaudible]?

Instructor (Christopher Manning):I mean, there aren't that many errors, but that is just the engram language model. And in particular, I think it's just a trigram language model. I mean, I can – yeah, to dwell on this for a moment, the example I showed earlier, I sort of said languages have these long-distance contexts and you need it to fully understand languages. That's true.

On the other hand, I mean, I think for a lot of linguists it's been a real eye-opener, or that they're still kind of in denial as to how well these engram language models work. Because the fact of the matter is, you know, there's no explicit representation of grammar whatsoever, but simply if you're learning up these engram probabilities, you're kind of inefficiently and badly, in some sense, because you're spewing around billions of parameters, but you're nevertheless, you're learning all the grammar as you go along.

That, you know, you learn that after a form of the verb "to be," you can get present participle forms of verbs – "I am singing –" and you can get passive participles – "I was hit –" that you don't get other forms. Like you don't get "I was make," all right? You just pick up all of that stuff as engrams. And so yeah, the thing to notice here is, well, actually, this is pretty grammatical stuff. And you should notice that too when you do the assignment, since one of the things in the assignment is to generate text.

And what you should find is by the time you have a decent trigram model that although the sentences will normally make not very much sense – and you know, quite a few of these don't make very much sense, either. Some of them are okay, but some of them don't make very much sense. They'll nevertheless come out sort of grammatical.

Okay, let me now get on to – back to MT. Yeah?

Student:[Inaudible] I guess in that model, sort of like for example, you see the word superconductivity [inaudible] realize whether or not there were three words [inaudible].

Instructor (Christopher Manning):Right.

Student: So there was [inaudible].

Instructor (Christopher Manning):So, I mean, I'm kind of guessing the example – well, I mean, obviously it can generate different things after superconductivity; that much is clear. I mean, an effect that it is worth being aware of is that these models clearly can be over-trained. So if they've only seen the word superconductivity a few times, even if you smooth, a large percentage of the probability mass will go to the words that you have seen.

And so if you then go off into a very technical topic, that these models will have a tendency to start reproducing a sequence of words that they saw verbatim, because essentially there are enough kind of very topic-specific words that nearly all the probability mass is just to sort of stay within that topic and reproduce the sentence verbatim.

And I think that's kind of what you're meaning, right? But nevertheless, it was a smoothed engram model, so it can sort of then move off and generate other words.

Okay, so we saw right in the first lecture this idea that we have a corpus of parallel text, and what we're going to want to do is use this corpus of parallel text in the same kind of data-driven way to learn translation models. And in particular, we're gonna wanna learn from these alignment models, so the little black bars are then alignment models which show which words translate which other words.

And our goal – and so we can think of those as having models of probability of French-given English. And if we put together a language model probability of English with a word translation model, probability of French-given English, we then have the two key ingredients for having a system that can translate from French to English.

And so this is what we want to build. So how are we gonna do that? The first part of it is, well, we need a lot of parallel data to be able to do this. And how can we get a lot of parallel data? Well, you know, if you know someone that speaks another language you can ask them to translate some sentences for you. That might get expensive if you want a lot of parallel data.

You can actually find lots of parallel data. I mean, it turns out that there are just an enormous number of companies that for various reasons produce parallel data, right? So software manufacturers will translate their manuals, as we discussed. Lots of other

equipment manufacturers will do the same. People who want to advertise in different languages.

And so you can kind of go out on the web and find parallel data, and if you want to work in not the most frequent languages, often, this is what you want to do. So if you want to build something like a Tagalog to English translator, pretty much the way to get parallel data is you have to start crawling the web, looking for places where there's parallel data.

For some of the commonest languages – but you have to do a lot of work this way, because there's all kinds of messy stuff in web pages. But for sort of well-resourced, big languages, a lot of the time what people use, certainly at universities and big corporations, is stuff from this place called the Linguistic Data Consortium, which hangs off the University of Pennsylvania and essentially they do a business of having big sets of data and distributing it.

And so for major language pairs like French, Arabic, Chinese – my graph is out of date, but there's sort of a hundred, 200 million words of parallel data that they've collected, and then you can just get on a handy DVD and have lots of material to build a translation model for.

There's also actually now – so this stuff, though, costs a little bit of money. Not a lot of time, but a little bit of money. There's actually now starting to be even some good free resources, and in particular, the best source of that is from the European Union. So that there are data sets from the European Union which are completely free, because a lot of the time government data isn't copyright and other people have done the work for you of aligning the different sentences and the different – in documents in the different languages. And that's actually the data that we'll use for the assignment.

Student:[Inaudible] the data, do they try to avoid putting in things that might be, like, [inaudible] translations where, you know [inaudible]?

Instructor (Christopher Manning):No. I mean, in all of this data collection, you're just putting in – well, for first approximation, you're just putting in everything. You're just putting in, you know, stuff you find. And although I'm saying that there's lots of material available, quite often, a lot of the material that's available is imperfect in many ways, in particular in terms of genre coverage.

So that the Chinese English, it's easy to get tons of data. It's all thanks to Hong Kong, all right? So they produce huge amounts of bilingual data through their legal system, their parliamentary system, their newspapers, etc. Well, first of all, there are dialect problems between different kinds of Chinese, but secondly, is a ton of law reports actually wonderful data if you want to be translating instant messaging conversations? Not really. It's the same problem of having the right data.

But yeah, I mean, it's just taking whatever sentences there is, partly because eventually what you wanna do is you wanna learn idiomatic translations. But to some extent, what

these statistical models end up being is kind of like example-based learning, and so that if you see examples of idioms to translate something, you can just learn that, and you will.

I mean, people do do a bit of data clean-up, but it's normally fairly crude. I mean, the kind of data clean-up that is done is it turns out that when people translate things, that commonly, entire sentences or entire clauses of sentences will disappear. You know, sometimes it's not clear whether they disappeared because the translator just goofed and forgot to do that bit, or whether the translator somehow made a decision that some of that detail wasn't important and just to abbreviate things.

But you'll certainly see these cases where the French sentence is 50 words long, and only the first clause of that was translated into English, and your English translation's only ten words long, and people try and clean up stuff like that because it's sort of perfectly obvious that most of the French just isn't being translated.

Okay, so from some source we've got a pile of stuff in one language and a pile of stuff in a second language. What do we need to do then? So we have to do some kind of data clean-up, because we're going to represent our sentences as sequences of tokens.

I mean, if you're doing English, that's not so hard. And even in English, it turns out there's a bunch of stuff that's not completely trivial, so you have to kind of separate off pieces of punctuation. Generally, you want to separate off and keep pieces of punctuation, because having the punctuation there actually helps you because it reveals some of the structure.

I mean, these ones sort of mainly seem obvious as to how to separate them off, but things get a bit more subtle with, you know, this period is useful, sorts of uses, decimal points, Internet, IP addresses, abbreviations, like et cetera, when it's not the end of a sentence. Hyphens are used for various things.

Some of the languages are as easy as English; other languages aren't as easy as English. Notably, east Asian languages don't put spaces between words.

Putting spaces between words is actually something that was invented in the Middle Ages, so if you go back, ancient Greek, as it was written by ancient Greeks, was like this. It wasn't written with spaces between the words. And so it was the Romans that initially had the idea of maybe you should do something to mark word boundaries, and so they'd chip a little dot in the stone between words, but not put any word spacing in. And so it was then really monks that had this idea that spaces between words would make things easier.

But people can read this stuff. Ancient Greeks and Chinese people. Okay, but normally, we want to have word-based models and so we need to do word segmentation, which is actually a task [inaudible] worked on quite a bit, and we'll come back to that kind of task later. For the moment, we'll just assume that we've chopped our Chinese text into words if we need to.

Okay, so we now have on both sides a sequence of words that are translations of each other, and so the next step is to work out how do these things translate each other? The first thing to know is these human translators, you'd think that if you gave them a sentence in one language, they'd just translate it into a sentence in the other language. I mean, is that too much to ask?

But it turns out that human translators don't do that. It is just quite regularly the case that they will combine two sentences in one language into one sentence in the other, or they'll decide a sentence is too long and awkward if it's translated as one sentence, and they'll split it into two sentences. And sometimes they even do more complex things.

They'll take a clause of one sentence and move it into the other sentence. And so, you have this first problem of doing sentence alignment, of working out which sentences are translations of which other sentences, including, as I mentioned, sentences are sometimes just dropped altogether.

I'm not really gonna talk about this. Essentially, you can do this as a dynamic programming problem. So this is essentially classic edit distance, right? So you can have substitutions, insertions, deletions, combining, you can do an edit distance problem and work it out, you can read about that if you want. But we are going to assume we already have sentence align data.

Okay, so this is then what we're building for our whole MT system. So as I introduced in the first class or second class, our goal for what we're going to do for the assignment is to build one of these noisy channel MT systems. So what we're doing is we're going to start with bilingual text, and we're going to learn a translation model of how words from one language translate into the other, and then right at the moment you should be working on assignment one, and we're also building an English language model.

And so if we put those two things together, we will be able to translate sentences. And so actually, what we're gonna give you is the stuff to put it together, so at the end of assignment two you should have a little MT system. Now your expectations shouldn't be so high as to how this MT system will work – probably not good enough for translating stuff you send home to your grandmother or something. But nevertheless, it should work and give you a little MT system.

Okay, and so the idea here is when we want to translate, we're going to be translating from Spanish to English, we'll start off with our Spanish sentence – the translation model will turn Spanish words into English words. And then we'll use the language model to put these English words into a good order. It sort of checks that we're getting kind of grammatical plausible output and can help choose between different translations.

Okay, so effectively, this use of a generative model [inaudible] seemed a neat idea because it allowed a division of labor so that the general idea is that we can use a translation model whose only job is to suggest good candidate translations. It says okay,

you've seen the word "ferme," here are possible translations for it: close, shut, finish. Those are possible translations, think about some of those.

And so it's only kind of doing how to translate a word, perhaps in context, but how to translate a word task and then we can have the English language model that says okay, given the stuff that you're translating, I can choose the words that go well in this context, and I can rearrange them in the good order based on the grammar rules of English.

So the language model actually knows that adjectives before nouns sound good in English, whereas adjectives after nouns sound really bad in English, even though they sound good in French and Spanish and other languages.

And so it was a kind of a division of the problem that seemed a useful way to do things, and so essentially all MT – all statistical MT work through about 2001 used this kind of generative model story. Modern work has actually moved away from that, and I'll talk a bit about that later, but you guys won't be doing it in terms of what you implement.

Okay. So we have to translate a model on the language model, so if we have these two things, the bit that we're going to provide to you that is left over is the tricky bit, effectively. The bit down here is normally called the decoder. Language and speech people talk about decoders and when they say decoders, what that means is search problem. Because all we've got is these two probability models and a Spanish sentence, and what we're wanting to do is say according to these two probability models, what is the English sentence that maximizes this score?

Turns out there [inaudible] a good way to find it, so actually this is an exponential search problem, and people use various heuristic methods to do a half-decent job at working out what's the best translation according to the model.

Okay, so for the rest of today, what I want to start talking about is how we take our parallel sentences – we've already sentence divided them – and build alignment models, and from those alignment models we then build probabilities of one word translating another word.

So this is the idea of an alignment. So we've got our sourced language sentenced – here it's German – and our English language sentence is up here. And essentially, what we're wanting to do, if you think of it as a grid like this, is put dots into this grid as to which words translate which words.

So yah, that translates as well. Ich, I. Think, right? We're sort of translating the words, and it starts off in the same word order so it goes up the diagonal. And then some more funny German stuff happens.

And our goal is to – what our goal in learning will be is to work out what those word alignments are, because if we know the word alignments we can then use these aligned

words to predict how to translate a word, and then we can get probability of foreign language word given English word.

Okay, so here's another way of visualizing alignments where you're just drawing lines between the two sentences. Let me just say for a couple of minutes what happens in the space of when you take sentences.

So it's just like at the sentence level if you think that you can translate languages word-for-word and have one word translate as one word, doesn't work out too well for you. So here we have French English. Japan's shaken by two new quakes. This is the really easy kind of case. If everything were this easy, we'd have really good statistical MT systems.

So here, basically every word translated as one word, and they occurred in the same order – no reordering. The only thing that happened was it turns out in French you put a definite article of "the" in front of country names. So this is [inaudible] Japan. So that's a kind of a word that came from nowhere in the French, and they're referred to in these MT models as spurious words – words that kind of have no source in the English.

We also get the opposite problem, and the program has been implemented. The program has been put in application sort of implementation. So here, the and in English didn't get translated at all. And so those words are referred to as zero fertility words, meaning they don't turn into anything in the other language.

And then here we have this trickier case where one English word got translated by this phrase in French, and so you can think of this word as having a fertility of three, and it produces three words in the other language.

But you keep on seeing different cases; you can not surprisingly get examples that go the other way. So here, we have multiple words in French are in English that are translated by one word in French. As I'll make clear in a minute, the models that we're going to develop actually can't handle this case, so the models that we're going to do, because of the way we set up the alignment models, have this annoying and linguistically inappropriate asymmetry that they can handle this case, they can handle an English word going to several French words, but they can't handle this case.

And so we just simulate this case by treating it as a one-to-one alignment and then say the other word doesn't translate – it's a zero fertility word. But that's obviously yucky, and not what's really happening languages.

Okay, and then here's kind of the M to N case, which is, I guess, the most complicated kind of alignment that you can get. These little kind of bits of aligned stuff in the original work on statistical machine translation that was done at IBM were referred to as sets. And other people commonly refer to them as beads or statistical phrases, or words like that.

Okay, so this is just the kind of comic view of how machine translation is going to work. We start with some parallel text, and we start thinking anything can translate anything.

But what we do is we look at our text and say hm, the word "the," maybe it's being translated as "la," maybe it's translated as "maison."

"House." Maybe it's being translated as "maison," maybe it's being translated as "la." But, well, look a bit more closely. If we said that "house" was being translated as "maison," we could make the probability of "maison" given "house" as high as we want. In fact, we can make it one for this data, because there's kind of nothing contesting with it.

And then we could also make the probability of "la" given "the" for this data at one, because there's be nothing contesting with that. So we can maximize the probability of the data if we assume that "la" and "the" are aligned, and "maison" and "house" are aligned, and so that's what we're going to do. We're going to use expectation maximization learning to implement this kind of pigeonhole idea where we gain value if we can translate words consistently.

Now of course words aren't translated consistently, so it doesn’t work as perfectly as this cartoon, but that's the idea of what's going to drive our learning.

Okay, that settles down. Okay. Okay, let me just summarize this important fact. Remember this so you're not completely confused. The confusing thing about this setup is what we're doing is we're taking a French sentence and wanting to translate it into an English sentence. If you were naïve and think you're doing that, you'd think hm, what I want is probability of French word given English word.

Note crucially that what we're going to estimate is exactly the opposite. We're going to estimate probability of French word given English word. Why is that? It's because we're playing this [inaudible] inversion game, so our model is that we've got the probability of English times the probability of French-given English, and then we're going to take the [inaudible] max over that product to find the most likely probability of English-given French.

So this is the opposite – these tables are the opposite way round to how we want to translate. That makes sense? It's easy to get confused on that point. I get confused every year at some point when I'm talking to somebody.

Okay, so the next section goes through these famous IBM models which were developed in the late '90s when they developed this sequence of better machine translation models. And we'll at least briefly go through all of them. But in the last few minutes, I want to introduce IBM model one.

So IBM model one is going to give these probabilities of a French sentence given an English sentence. There's a set of English words and the extra English word "null," so "null" would be the source of the spurious words, which don't actually come from any English words.

And so what we're wanting to do is we're wanting to work out a probability of a French sentence given an English sentence. And we're going to do that in terms of assuming a [inaudible] alignment, so this is where this EM idea is going to come in.

Our hidden data is how the English and French words align to each other. So we're going to say the probability of French-given English is the sum over all possible alignments – exponential number of those – of the probability of French and the alignment given the English.

And so at that point, we can then break this apart and work it out in terms of individual words. In particular, model one makes some very strong assumptions which allows it to be very easy to estimate, and in fact makes it not exponential anymore. So let us assume that every one of these alignments is independent from each other.

That makes things easy. So now we can just say it's a product over J of each one of these alignments. Okay, so how are we going to represent alignments if we're going to have these A variables? And the A variables say AJ equals I, so this – what we do is say that A5 equals six. And so that corresponds to this line. A3 equals four, that corresponds to this line.

So the variable is saying what English word each French word is aligned to, and this is exactly where you get this asymmetry immediately. But because you have this representation of the alignment variables, you can have multiple French swords aligned to one English word, but you just can't represent the reverse, right? Because the value of – this is a variable with a value, and its value is a number that is some English word.

Okay, so if the probability of a French sentence with a particular alignment to the English sentence is that you just – you've got the alignment, you've got the French sentence, you just take the product of each one of these where you take the probability that AJ equals I times the translation probability of what's the probability FJ given EI. Okay.

Then to make model one super-super-simple, we make one further assumption. Our next assumption is we say let's suppose we have no substantive model over which alignments occur. So we're just going to put a uniform distribution over possible alignments. So the probability of AJ equals I is just going to be uniform. And so you can work it out in terms of the length of the sentence as a uniform distribution.

So the probabilities are just in terms of this, and this term essentially just goes away, because since it's uniform you don't actually have to keep it around, because it's not going to be changing what you're choosing.

And so that then leads to this slide here, which I will just put up. Yeah, I think it's probably more sensible if I go through this slide right at the beginning of next time, but I will nevertheless point it out. So for the first part of the assignment, what you do is have to do model one, and then you go on and do model two.

For model one, it is really the case that if you just take this slide and frame it and stick it above your monitor and implement this, your model one will be correct and work. But if you want to be a thoughtful person, you should do a little bit more work and actually read the Kevin Knight material and actually work out why this is correct.

This is correct, you can do this and it works. But if you actually start thinking about it in the model, it's not immediately obvious in terms of how the model is presented as in Kevin Knight as to why this is correct. It essentially ends up being correct that because of the fact that all the alignments are uniform and the sentence length you don't have to consider and various things that all the other stuff goes away, and in terms of the estimation equation, this equation here works. But I do encourage you to actually sort of read through and try and understand why some of this works.

Okay, so there aren't any questions up to here. I'll stop here now, and then next time I'll start with going through model one, go through the various IBM models.

[End of Audio]

Duration: 75 minutes