PETER SZOLOVITS: All right. Let's get started. Good afternoon. So last time, I started talking about the use of natural language processing to process clinical data. And things went a little bit slowly. And so we didn't get through a lot of the material. I'm going to try to rush a bit more today. And as a result, I have a lot of stuff to cover.

So if you remember, last time, I started by saying that a lot of the NLP work involves coming up with phrases that one might be interested in to help identify the kinds of data that you want, and then just looking for those in text. So that's a very simple method. But it's one that works reasonably well. And then Kat Liao was here to talk about some of the applications of that kind of work in what she's been doing in cohort selection.

So what I want to talk about today is more sophisticated versions of that, and then move on to more contemporary approaches to natural language processing. So this is a paper that was given to you as one of the optional readings last time. And it's work from David Sontag's lab, where they said, well, how do we make this more sophisticated?

So they start the same way. They say, OK, Dr. Liao, let's say, give me terms that are very good indicators that I have the right kind of patient, if I find them in the patient's notes. So these are things with high predictive value. So you don't want to use a term like sick, because that's going to find way too many people. But you want to find something that is very specific but that has a high predictive value that you are going to find the right person. And then what they did is they built a model that tries to predict the presence of that word in the text from everything else in the medical record.

So now, this is an example of a silver-standard way of training a model that says, well, I don't have the energy or the time to get doctors to look through thousands and thousands of records. But if I select these anchors well enough, then I'm going to get a high yield of correct responses from those. And then I train a machine learning model that learns to identify those same terms, or those same records that have those terms in them. And by the way, from that, we're going to learn a whole bunch of other terms that are proxies for the ones that we started with. So this is a way of enlarging that set of terms automatically.

And so there are a bunch of technical details that you can find out about by reading the paper. They used a relatively simple representation, which is essentially a bag-of-words

representation. They then sort of masked the three words around the word that actually is the one they're trying to predict just to get rid of short-term syntactic correlations.

And then they built an L2-regularized logistic regression model that said, what are the features that predict the occurrence of this word? And then they expanded the search vocabulary to include those features as well. And again, there are tons of details about how to discretize continuous values and things like that that you can find out about.

So you build a phenotype estimator from the anchors and the chosen predictors. They calculated a calibration score for each of these other predictors that told you how well it predicted. And then you can build a joint estimator that uses all of these. And the bottom line is that they did very well.

So in order to evaluate this, they looked at eight different phenotypes for which they had human judgment data. And so this tells you that they're getting AUCs of between 0.83 and 0.95 for these different phenotypes. So that's quite good.

They, in fact, were estimating not only these eight phenotypes but 40-something. I don't remember the exact number, much larger number. But they didn't have validated data against which to test the others. But the expectation is that if it does well on these, it probably does well on the others as well.

So this was a very nice idea. And just to illustrate, if you start with something like diabetes as a phenotype and you say, well, I'm going to look for anchors that are a code of 250 diabetes mellitus, or I'm going to look at medication history for diabetic therapy-- so those are the silver-standard goals that I'm looking at. And those, in fact, have a high predictive value for somebody being in the cohort. And then they identify all these other features that predict those, and therefore, in turn, predict appropriate selectors for the phenotype that they're interested in.

And if you look at the paper again, what you see is that this outperforms, over time, the standard supervised baseline that they're comparing against, where you're getting much higher accuracy early in a patient's visit to be able to identify them as belonging to this cohort. I'm going to come back later to look at another similar attempt to generalize from a core using a different set of techniques. So you should see that in about 45 minutes, I hope.

Well, context is important. So if you look at a sentence like Mr. Huntington was treated for

Huntington's disease at Huntington Hospital, located on Huntington Avenue, each of those mentions of the word Huntington is different. And for example, if you're interested in eliminating personally identifiable health information from a record like this, then certainly you want to get rid of the Mr. Huntington part.

You don't want to get rid of Huntington's disease, because that's a medically relevant fact. And you probably do want to get rid of Huntington Hospital and its location on Huntington Avenue, although those are not necessarily something that you're prohibited from retaining. So for example, if you're trying to do quality studies among different hospitals, then it would make sense to retain the name of the hospital, which is not considered identifying of the individual.

So we, in fact, did a study back in the mid 2000s, where we were trying to build an improved de-identifier. And here's the way we went about it. This is a kind of kitchen sink approach that says, OK, take the text, tokenize it. Look at every single token. And derive things from it.

So the words that make up the token, the part of speech, how it's capitalized, whether there's punctuation around it, which document section is it in-- many databases have sort of conventional document structure. If you've looked at the mimic discharge summaries, for example, there's a kind of prototypical way in which that flows from beginning to end. And you can use that structural information.

We then identified a bunch of patterns and thesaurus terms. So we looked up, in the UMLS, words and phrases to see if they matched some clinically meaningful term. We had patterns that identified things like phone numbers and social security numbers and addresses and so on.

And then we did parsing of the text. So in those days, we used something called the Link Grammar Parser, which, doesn't make a whole lot of difference what parser. But you get either a constituent or constituency or dependency parse, which gives you relationships among the words. And so it allows you to include, as features, the way in which a word that you're looking at relates to other words around it.

And so what we did is we said, OK, the lexical context includes all of the above kind of information for all of the words that are either literally adjacent or within n words of the original word that you're focusing on, or that are linked by within k links through the parse to that word. So this gives you a very large set of features. And of course, parsing is not a solved problem. And so this is an example from that story that I showed you last time. And if you see, it comes

up with 24 ambiguous parses of this sentence. So there are technical problems about how to deal with that.

Today, you could use a different parser. The Stanford Parser, for example, probably does a better job than the one we were using 14 years ago and gives you at least more definitive answers. And so you could use that instead.

And so if you look at what we did, we said, well, here is the text "Mr." And here are all the ways that you can look it up in the UMLS. And it turns out to be very ambiguous. So M-R stands not only for mister, but it also stands for Magnetic Resonance. And it stands for a whole bunch of other things. And so you get huge amounts of ambiguity.

"Blind" turns out also to give you various ambiguities. So it maps here to four different concept-unique identifiers. "Is" is OK. "79-year-old" is OK. And then "male," again, maps to five different concept-unique identifiers. So there are all these problems of over-generation from this database. And here's some more, but I'm going to skip over that.

And then the learning model, in our case, was a support vector machine for this project, in which we just said, well, throw in all the-- you know, it's the kill them all, and God will sort them out kind of approach. So we just threw in all these features and said, oh, support vector machines are really good at picking out exactly what are the best features. And so we just relied on that.

And sure enough, so you wind up with literally millions of features. But sure enough, it worked pretty well. And so Stat De-ID was our program. And you see that on real discharge summaries, we're getting precision and recall on PHI up around 98 and 1/2%, 95 and 1/4%, which was much better than the previous state of the art, which had been based on rules and dictionaries as a way of de-identifying things.

So this was a successful example of that approach. And of course, this is usable not only for de-identification. But it's also usable for entity recognition. Because instead of selecting entities that are personally identifiable health information, you could train it to select entities that are diseases or that are medications or that are various other things.

And so this was, in the 2000s, a pretty typical way for people to approach these kinds of problems. And it's still used today. There are tools around that let you do this. And they work reasonably effectively. They're not state of the art at the moment, but they're simpler than

many of today's state of the art methods.

So here's another approach. This was something we published a few years ago, where we started working with some psychiatrists and said, could we predict 30-day readmission for a psychiatric patient with any degree of reliability? That's a hard prediction. Willie is currently running an experiment where we're asking psychiatrists to predict that. And it turns out, they're barely better than chance at that prediction. So it's not an easy task.

And what we did is we said, well, let's use topic modeling. And so we had this cohort of patients, close to 5,000 patients. About 10% of them were readmitted with a psych diagnosis. And almost 3,000 of them were readmitted with other diagnoses.

So one thing this tells you right away is that if you're dealing with psychiatric patients, they come and go to the hospital frequently. And this is not good for the hospital's bottom line because of reimbursement policies of insurance companies and so on. So of the 4,700, only 1,240 were not readmitted within 30 days. So there's very frequent bounce-back.

So we said, well, let's try building a baseline model using a support vector machine from baseline clinical features like age, gender, public health insurance as a proxy for socioeconomic status. So if you're on Medicaid, you're probably poor. And if you have private insurance, then you're probably an MIT employee and/or better off. So that's a frequently used proxy, a comorbidity index that tells you sort of how sick you are from things other than your psychiatric problems.

And then we said, well, what if we add to that model common words from notes. So we said, let's do a TF-IDF calculation. So this is term frequency divided by log of the document frequency. So it's sort of, how specific is a term to identify a particular kind of condition? And we take the 1,000 most informative words, and so there are a lot of these.

So if you use 1,000 most informative words from these nearly 5,000 patients, you wind up with something like 66,000 words, unique words, that are informative for some patient. But if you limit yourself to the top 10, then it only uses 18,000 words. And if you limit yourself to the top one, then it uses about 3,000 words.

And then we said, well, instead of doing individual words, let's do a latent Dirichlet allocation. So topic modeling on all of the words, as a bag of words-- so no sequence information, just the collection of words. And so we calculated 75 topics from using LDA on all these notes.

So just to remind you, the LDA process is a model that says every document consists of a certain mixture of topics, and each of those topics probabilistically generates certain words. And so you can build a model like this, and then solve it using complicated techniques. And you'd wind up with topics, in this study, as follows. I don't know. Can you read these? They may be too small.

So these are unsupervised topics. And if you look at the first one, it says patient, alcohol, withdrawal, depression, drinking, and Ativan, ETOH, drinks, medications, clinic inpatient, diagnosis, days, hospital, substance, use treatment program, name. That's a de-identified use/abuse problem number. And we had our experts look at these topics. And they said, oh, well, that topic is related to alcohol abuse, which seems reasonable.

And then you see, on the bottom, psychosis, thought features, paranoid psychosis, paranoia symptoms, psychiatric, et cetera. And they said, OK, that's a psychosis topic. So in retrospect, you can assign meaning to these topics. But in fact, they're generated without any a priori notion of what they ought to be. They're just a statistical summarization of the common co-occurrences of words in these documents.

But what you find is that if you use the baseline model, which used just the demographic and clinical variables, and you say, what's the difference in survival, in this case, in time to readmission between one set and another in this cohort, and the answer is they're pretty similar. Whereas, if you use a model that predicts based on the baseline and 75 topics, the 75 topics that we identified, you get a much bigger separation. And of course, this is statistically significant. And it tells you that this technique is useful for being able to improve the prediction of a cohort that's more likely to be readmitted from a cohort that's less likely to be readmitted.

It's not a terrific prediction. So the AUC for this model was only on the order of 0.7. So you know, it's not like 0.99. But nevertheless, it provides useful information.

The same group of psychiatrists that we worked with also did a study with a much larger cohort but much less rich data. So they got all of the discharges from two medical centers over a period of 12 years. So they had 845,000 discharges from 458,000 unique individuals. And they were looking for suicide or other causes of death in these patients to see if they could predict whether somebody is likely to try to harm themselves, or whether they're likely to die accidentally, which sometimes can't be distinguished from suicide.

So the censoring problems that David talked about are very much present in this. Because you lose track of people. It's a highly imbalanced data set. Because out of the 845,000 patients, only 235 committed suicide, which is, of course, probably a good thing from a societal point of view but makes the data analysis hard. On the other hand, all-cause mortality was about 18% during nine years of a follow-up. So that's not so imbalanced.

And then what they did is they curated a list of 3,000 terms that correspond to what, in the psychiatric literature, is called positive valence. So this is concepts like joy and happiness and good stuff, as opposed to negative valence, like depression and sorrow and all that stuff. And they said, well, we can use these types of terms in order to help distinguish among these patients.

And what they found is that, if you plot the Kaplan-Meier curve for different quartiles of risk for these patients, you see that there's a pretty big difference between the different quartiles. And you can certainly identify the people who are more likely to commit suicide from the people who are less likely to do so. This curve is for suicide or accidental death. So this is a much larger data set, and therefore the error bars are smaller. But you see the same kind of separation here. So these are all useful techniques.

Now I'll to another approach. This was work by one of my students, Yuon Wo, who was working with some lymphoma pathologists at Mass General. And so the approach they took was to say, well, if you read a pathology report about somebody with lymphoma, can we tell what type of lymphoma they had from the pathology report if we blank out the part of the pathology report that says, "I, the pathologist, think this person has non-Hodgkin's lymphoma," or something? So from the rest of the context, can we make that prediction?

Now, Yuon took a kind of interesting, slightly odd approach to it, which is to treat this as an unsupervised learning problem rather than as a supervised learning problem. So he literally masked the real answer and said, if we just treat everything except what gives away the answer as just data, can we essentially cluster that data in some interesting way so that we re-identify the different types of lymphoma?

Now, the reason this turns out to be important is because lymphoma pathologists keep arguing about how to classify lymphomas. And every few years, they revise the classification rules. And so part of his objective was to say, let's try to provide an unbiased, data-driven method that may help identify appropriate characteristics by which to classify these different

lymphomas.

So his approach was a tensor factorization approach. You often see data sets like this that's, say, patient by a characteristic. So in this case, laboratory measurements-- so systolic/diastolic blood pressure, sodium, potassium, et cetera. That's a very vanilla matrix encoding of data. And then if you add a third dimension to it, like this is at the time of admission, 30 minutes later, 60 minutes later, 90 minutes later, now you have a three-dimensional tensor.

And so just like you can do matrix factorization, as in the picture above, where we say, my matrix of data, I'm going to assume is generated by a product of two matrices, which are smaller in dimension. And you can train this by saying, I want entries in these two matrices that minimize the reconstruction error. So if I multiply these matrices together, then I get back my original matrix plus error. And I want to minimize that error, usually root mean square, or mean square error, or something like that.

Well, you can play the same game for a tensor by having a so-called core tensor, which identifies the subset of characteristics that subdivide that dimension of your data. And then what you do is the same game. You have matrices corresponding to each of the dimensions. And if you multiply this core tensor by each of these matrices, you reconstruct the original tensor. And you can train it again to minimize the reconstruction loss.

So there are, again, a few more tricks. Because this is dealing with language. And so this is a typical report from one of these lymphoma pathologists that says immunohistochemical stains show that the follicles-- blah, blah, blah, blah, blah-- so lots and lots of details. And so he needed a representation that could be put into this matrix tensor, this tensor factorization form.

And what he did is to say, well, let's see. If we look at a statement like this, immuno stains show that large atypical cells are strongly positive for CD30, negative for these other surface expressions. So the sentence tells us relationships among procedures, types of cells, and immunologic factors.

And for feature choice, we can use words. Or we can use UMLS concepts. Or we can find various kinds of mappings. But he decided that in order to retain the syntactic relationships here, what he would do is to use a graphical representation that came out of, again, parsing all of these sentences.

And so what you get is that this creates one graph that talks about the strongly positive for

CD30, large atypical cells, et cetera. And then you can factor this into subgraphs. And then you also have to identify frequently occurring subgraphs. So for example, large atypical cells appears here, and also appears there, and of course will appear in many other places. Yeah?

AUDIENCE: Is this parsing domain in language diagnostics? For example, did they incorporate some sort of medical information here or some sort of linguistic--

PETER SZOLOVITS: So in this particular study, he was using the Stanford Parser with some tricks. So the Stanford Parser doesn't know a lot of the medical words. And so he basically marked these things as noun phrases.

And then the Stanford Parser also doesn't do well with long lists like the set of immune features. And so he would recognize those as a pattern, substitute a single made-up word for them, and that made the parser work much better on it. So there were a whole bunch of little tricks like that in order to adapt it. But it was not a model trained specifically on this. I think it's trained on *Wall Street Journal* corpus or something like that. So it's general English.

AUDIENCE: Those are things that he did manually as opposed to, say, [INAUDIBLE]?

PETER SZOLOVITS: No. He did it algorithmically, but he didn't learn which algorithms to use. He made them up by hand.

But then, of course, it's a big corpus. And he ran these programs over it that did those transformations. So he calls it two-phase parsing. There's a reference to his paper on the first slide in this section if you're interested in the details. It's described there.

So what he wound up with is a tensor that has patients on one axis, the words appearing in the text on another axis. So he's still using a bag-of-words representation. But the third axis is these language concept subgraphs that we were talking about. And then he does tensor factorization on this.

And what's interesting is that it works much better than I expected. So if you look at his technique, which he called SANTF, the precision and recall are about 0.72 and 0.854 macro-average and 0.754 micro-average, which is much better than the non-negative matrix factorization results, which only use patient by word or patient by subgraph, or, in fact, one where you simply do patient and concatenate the subgraphs and the words in one dimension. So that means that this is actually taking advantage of the three-way relationship.

If you read papers from about 15, 20 years ago, people got very excited about the idea of bi-clustering, which is, in modern terms, the equivalent of matrix factorization. So it says given two dimensions of data, and I want to cluster things, but I want to cluster them in such a way that the clustering of one dimension helps the clustering of the other dimension. So this is a formal way of doing that relatively efficiently. And tensor factorization is essentially tri-clustering.

So now I'm going to turn to the last of today's big topics, which is language modeling. And this is really where the action is nowadays in natural language processing in general. I would say that the natural language processing on clinical data is somewhat behind the state of the art in natural language processing overall.

There are fewer corpora that are available. There are fewer people working on it. And so we're catching up. But I'm going to lead into this somewhat gently.

So what does it mean to model a language? I mean, you could imagine saying it's coming up with a set of parsing rules that define the syntactic structure of the language. Or you could imagine saying, as we suggested last time, coming up with a corresponding set of semantic rules that say a concept or terms in the language correspond to certain concepts and that they are a combinatorially, functionally combined as the syntax directs, in order to give us a semantic representation.

So we don't know how to do either of those very well. And so the current, the contemporary idea about language modeling is to say, given a sequence of tokens, predict the next token. If you could do that perfectly, presumably you would have a good language model. So obviously, you can't do it perfectly. Because we don't always say the same word after some sequence of previous words when we speak. But probabilistically, you can get close to that.

And there's usually some kind of Markov assumption that says that the probability of emitting a token given the stuff that came before it is ordinarily dependent only on n previous words rather than on all of history, on everything you've ever said before in your life. And there's a measure called perplexity, which is the entropy of the probability distribution over the predicted words. And roughly speaking, it's the number of likely ways that you could continue the text if all of the possibilities were equally likely.

So perplexity is often used, for example, in speech processing. We did a study where we were trying to build a speech system that understood a conversation between a doctor and a

patient. And we ran into real problems, because we were using software that had been developed to interpret dictation by doctors. And that was very well trained. But it turned out-- we didn't know this when we started-- that the language that doctors use in dictating medical notes is pretty straightforward, pretty simple. And so it's perplexity is about nine, whereas conversations are much more free flowing and cover many more topics. And so its perplexity is about 73.

And so the model that works well for perplexity nine doesn't work as well for perplexity 73. And so what this tells you about the difficulty of accurately transcribing speech is that it's hard. It's much harder. And that's still not a solved problem.

Now, you probably all know about Zipf's law. So if you empirically just take all the words in all the literature of, let's say, English, what you discover is that the n-th word is about one over n as probable as the first word. So there is a long-tailed distribution.

One thing you should realize, of course, is if you integrate one over n from zero to infinity, it's infinite. And that may not be an inaccurate representation of language, because language is productive and changes. And people make up new words all the time and so on. So it may actually be infinite.

But roughly speaking, there is a kind of decline like this. And interestingly, in the brown corpus, the top 10 words make up almost a quarter of the size of the corpus. So you write a lot of thes, ofs, ands, a's, twos, ins, et cetera, and much less hematemesis, obviously.

So what about n-gram models? Well, remember, if we make this Markov assumption, then all we have to do is pay attention to the last n tokens before the one that we're interested in predicting. And so people have generated these large corpora n-grams.

So for example, somebody, a couple of decades ago, took all of Shakespeare's writings-- I think they were trying to decide whether he had written all his works or whether the earl of somebody or other was actually the guy who wrote Shakespeare. You know about this controversy? Yeah.

So that's why they were doing it. But anyway, they created this corpus. And they said-- so Shakespeare had a vocabulary of about 30,000 words and about 300,000 bigrams, and out of 844 million possible bigrams. So 99.96% of bigrams were never seen. So there's a certain regularity to his production of language.

Now, Google, of course, did Shakespeare one better. And they said, hmm, we can take a terabyte corpus-- this was in 2006. I wouldn't be surprised if it's a petabyte corpus today. And they published this. They just made it available. So there were 13.6 million unique words that occurred at least 200 times in this tera-word corpus. And there were 1.2 billion five-word sequences that occurred at least 40 times.

So these are the statistics. And if you're interested, there's a URL. And here's a very tiny part of their database. So ceramics, collectibles, collectibles-- I don't know-- occurred 55 times in a terabyte of text. Ceramics collectibles fine, ceramics collectibles by, pottery, cooking, comma, period, end of sentence, and, at, is, et cetera-- different number of times. Ceramics comes from occurred 660 times, which is reasonably large number compared to some of its competitors here.

If you look at four-grams, you see things like serve as the incoming, blah, blah, blah, 92 times; serve as the index, 223 times; serve as the initial, 5,300 times. So you've got all these statistics.

And now, given those statistics, we can then build a generator. So we can say, all right. Suppose I start with the token, which is the beginning of a sentence, or the separator between sentences. And I say sample a random bigram starting with the beginning of a sentence and a word, according to its probability, and then sample the next bigram from that word and all the other words, according to its probability, and keep doing that until you hit the end of sentence marker.

So for example, here I'm generating the sentence, I, starts with I, then followed by want, followed by two, followed by get, followed by Chinese, followed by food, followed by end of sentence. So I've just generated, "I want to get Chinese food," which sounds like a perfectly good sentence.

So here's what's interesting. If you look back again at the Shakespeare corpus and saying, if we generated Shakespeare from unigrams, you get stuff like at the top, "To him swallowed confess here both. Which. Of save on trail for are ay device and rote life have." It doesn't sound terribly good. It's not very grammatical. It doesn't have that sort of Shakespearean English flavor. Although, you do have words like nave and ay and so on that are vaguely reminiscent.

Now, if you go to bigrams, it starts to sound a little better. "What means, sir. I confess she? Then all sorts, he is trim, captain." That doesn't make any sense. But it starts to sound a little better.

And with trigrams, we get, "Sweet prince, Falstaff shall die. Harry of Monmouth," et cetera. So this is beginning to sound a little Shakespearean.

And if you go to quadrigrams, you get, "King Henry. What? I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in," et cetera. I mean, when I first saw this, like 20 years ago or something, I was stunned. This is actually generating stuff that sounds vaguely Shakespearean and vaguely English-like.

Here's an example of generating the *Wall Street Journal.* So from unigrams, "Months the my and issue of year foreign new exchanges September were recession." It's word salad.

But if you go to trigrams, "They also point to ninety nine point six billion from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil." So you could imagine that this is some *Wall Street Journal* writer on acid writing this text. Because it has a little bit of the right kind of flavor.

So more recently, people said, well, we ought to be able to make use of this in some systematic way to help us with our language analysis tasks. So to me, the first effort in this direction was Word2Vec, which was Mikolov's approach to doing this. And he developed two models.

He said, let's build a continuous bag-of-words model that says what we're going to use is co-occurrence data on a series of tokens in the text that we're trying to model. And we're going to use a neural network model to predict the word from the words around it. And in that process, we're going to use the parameters of that neural network model as a vector. And that vector will be the representation of that word.

And so what we're going to find is that words that tend to appear in the same context will have similar representations in this high-dimensional vector. And by the way, high-dimensional, people typically use like 300 or 500 dimensional vectors. So there's a lot of-- it's a big space.

And the words are scattered throughout this. But you get this kind of cohesion, where words that are used in the same context appear close to each other. And the extrapolation of that is that if words are used in the same context, maybe they share something about meaning.

So the other model is a skip-gram model, where you're doing the prediction in the other direction. From a word, you're predicting the words that are around it. And again, you are using a neural network model to do that. And you use the parameters of that model in order to represent the word that you're focused on.

So what came as a surprise to me is this claim that's in his original paper, which is that not only do you get this effect of locality as corresponding meaning but that you get relationships that are geometrically represented in the space of these embeddings. And so what you see is that if you take the encoding of the word man and the word woman and look at the vector difference between them, and then apply that same vector difference to king, you get close to queen. And if you apply it uncle, you get close to aunt.

And so they showed a number of examples. And then people have studied this. It doesn't hold it perfectly well. I mean, it's not like we've solved the semantics problem. But it is a genuine relationship.

The place where it doesn't work well is when some of these things are much more frequent than others. And so one of the examples that's often cited is if you go, London is to England as Paris is to France, and that one works. But then you say as Kuala Lumpur is to Malaysia, and that one doesn't work so well. And then you go, as Juba or something is to whatever country it's the capital of. And since we don't write about Africa in our newspapers, there's very little data on that. And so that doesn't work so well.

So there was this other paper later from van der Maaten and Geoff Hinton, where they came up with a visualization method to take these high-dimensional vectors and visualize them in two dimensions. And what you see is that if you take a bunch of concepts that are count concepts-- so 1/2, 30, 15, 5, 4, 2, 3, several, some, many, et cetera-- there is a geometric relationship between them. So they, in fact, do map to the same part of the space. Similarly, minister, leader, president, chairman, director, spokesman, chief, head, et cetera form a kind of cluster in the space. So there's definitely something to this.

I promised you that I would get back to a different attempt to try to take a core of concepts that you want to use for term-spotting and develop an automated way of enlarging that set of concepts in order to give you a richer vocabulary by which to try to identify cases that you're interested in. So this was by some of my colleagues, including Kat, who you saw on Tuesday.

And they said, well, what we'd like is the fully automated and robust, unsupervised feature selection method that leverages only publicly available medical knowledge sources instead of VHR data.

So the method that David's group had developed, which we talked about earlier, uses data from electronic health records, which means that you move to different hospitals and there may be different conventions. And you might imagine that you have to retrain that sort of method, whereas here the idea is to derive these surrogate features from knowledge sources. So unlike that earlier model, here they built a Word2Vec skip-gram model from about 5 million Springer articles-- so these are published medical articles-- to yield 500 dimensional vectors for each word.

And then what they did is they took the concept names that they were interested in and their definitions from the UMLS, and then they summoned the word vectors for each of these words, weighted by inverse document frequency. So it's sort of a TF-IDF-like approach to weight different words. And then they went out and they said, OK, for every disease that's mentioned in Wikipedia, Medscape, eMedicine, the Merck Manuals Professional Edition, the Mayo Clinic Diseases and Conditions, MedlinePlus Medical Encyclopedia, they used named entity recognition techniques to find all the concepts that are related to this phenotype.

So then they said, well, there's a lot of randomness in these sources, and maybe in our extraction techniques. But if we insist that some concept appear in at least three of these five sources, then we can be pretty confident that it's a relevant concept. And so they said, OK, we'll do that. Then they chose the top k concepts whose embedding vectors are closest by cosine distance to the embedding of this phenotype that they've calculated. And they say, OK, the phenotype is going to be a linear combination of all these related concepts.

So again, this is a bit similar to what we saw before. But here, instead of extracting the data from electronic medical records, they're extracting it from published literature and these web sources. And again, what you see is that the expert-curated features for these five phenotypes, which are coronary artery disease, rheumatoid arthritis, Crohn's disease, ulcerative colitis, and pediatric pulmonary arterial hypertension, they started with 20 to 50 curated features. So these were the ones that the doctors said, OK, these are the anchors in David's terminology. And then they expanded these to a larger set using the technique that I just described, and then selected down to the top n that were effective in finding relevant phenotypes.

And this is a terrible graph that summarizes the results. But what you're seeing is that the orange lines are based on the expert-curated features. This is based on an earlier version of trying to do this. And SEDFE is the technique that I've just described. And what you see is that the automatic techniques for many of these phenotypes are just about as good as the manually curated ones. And of course, they require much less manual curation. Because they're using this automatic learning approach.

Another interesting example to return to the theme of de-identification is a couple of my students, a few years ago, built a new de-identifier that has this rather complicated architecture. So it starts with a bi-directional recursive neural network model that is implemented over the character sequences of words in the medical text. So why character sequences? Why might those be important?

Well, consider a misspelled word, for example. Most of the character sequence is correct. There will be a bug in it at the misspelling. Or consider that a lot of medical terms are these compound terms, where they're made up of lots of pieces that correspond to Greek or Latin roots. So learning those can actually be very helpful.

So you start with that model. You then could concatenate the results from both the left-running and the right-running recursive neural network. And concatenate that with the Word2Vec embedding of the whole word. And you feed that into another bi-directional RNN layer.

And then for each word, you take the output of those RNNs, run them through a feed-forward neural network in order to estimate the prob-- it's like a soft max. And you estimate the probability of this word belonging to a particular category of personally identifiable health information. So is it a name? Is it an address? Is it a phone number? Is it or whatever?

And then the top layer is a kind of conditional random field-like layer that imposes a sequential probability distribution that says, OK, if you've seen a name, then what's the next most likely thing that you're going to see? And so you combine that with the probability distributions for each word in order to identify the category of PHI or non-PHI for that word. And this did insanely well.

So optimized by F1 score, we're up at a precision of 99.2%, recall of 99.3%. Optimized by recall, we're up at about 98%, 99% for each of them. So this is doing quite well.

Now, there is a non-machine learning comment to make, which is that if you read the HIPAA

law, the HIPAA regulations, they don't say that you must get rid of 99% of the personally identifying information in order to be able to share this data for research. It says you have to get rid of all of it. So no technique we know is 100% perfect. And so there's a kind of practical understanding among people who work on this stuff that nothing's going to be perfect. And therefore, that you can get away with a little bit. But legally, you're on thin ice.

So I remember many years ago, my wife was in law school. And I asked her at one point, so what can people sue you for? And she said, absolutely anything. They may not win. But they can be a real pain if you have to go defend yourself in court.

And so this hasn't played out yet. We don't know if a de-identifier that is 99% sensitive and 99% specific will pass muster with people who agree to release data sets. Because they're worried, too, about winding up in the newspaper or winding up getting sued.

Last topic for today-- so if you read this interesting blog, which, by the way, has a very good tutorial on BERT, he says, "The year 2018 has been an inflection point for machine learning models handling text, or more accurately, NLP. Our conceptual understanding of how best to represent words and sentences in a way that best captures underlying meanings and relationships is rapidly evolving."

And so there are a whole bunch of new ideas that have come about in about the last year or two years, including ELMo, which learns context-specific embeddings, the Transformer architecture, this BERT approach. And then I'll end with just showing you this gigantic GPT model that was developed by the OpenAI people, which does remarkably better than the stuff I showed you before in generating language.

All right. If you look inside Google Translate, at least as of not long ago, what you find is a model like this. So it's essentially an LSTM model that takes input words and munges them together into some representation, a high-dimensional vector representation, that summarizes everything that the model knows about that sentence that you've just fed it. Obviously, it has to be a pretty high-dimensional representation, because your sentence could be about almost anything. And so it's important to be able to capture all that in this representation.

But basically, at this point, you start generating the output. So if you're translating English to French, these are English words coming in, and these are French words going out, in sort of the way I showed you, where we're generating Shakespeare or we're generating *Wall Street*

Journal text. But the critical feature here is that in the initial version of this, everything that you learned about this English sentence had to be encoded in this one vector that got passed from the encoder into the decoder, or from the source language into the target language generator.

So then someone came along and said, hmm-- someone, namely these guys, came along and said, wouldn't it be nice if we could provide some auxiliary information to the generator that said, hey, which part of the input sentence should you pay attention to? And of course, there's no fixed answer to that. I mean, if I'm translating an arbitrary English sentence into an arbitrary French sentence, I can't say, in general, look at the third word in the English sentence when you're generating the third word in the French sentence. Because that may or may not be true, depending on the particular sentence.

But on the other hand, the intuition is that there is such a positional dependence and a dependence on what the particular English word was that is an important component of generating the French word. And so they created this idea that in addition to passing along the this vector that encodes the meaning of the entire input and the previous word that you had generated in the output, in addition, we pass along this other information that says, which of the input words should we pay attention to? And how much attention should we pay to them? And of course, in the style of these embeddings, these are all represented by high-dimensional vectors, high-dimensional real number vectors that get combined with the other vectors in order to produce the output.

Now, a classical linguist would look at this and retch. Because this looks nothing like classical linguistics. It's just numerology that gets trained by stochastic gradient descent methods in order to optimize the output. But from an engineering point of view, it works quite well.

So then for a while, that was the state of the art. And then last year, these guys, Vaswani et al. came along and said, you know, we now have this complicated architecture, where we are doing the old-style translation where we summarize everything into one vector, and then use that to generate a sequence of outputs. And we have this attention mechanism that tells us how much of various inputs to use in generating each element of the output. Is the first of those actually necessary?

And so they published this lovely paper saying attention is all you need, that says, hey, you know that thing that you guys have added to this translation model. Not only is it a useful addition, but in fact, it can take the place of the original model. And so the Transformer is an

architecture that is the hottest thing since sliced bread at the moment, that says, OK, here's what we do.

We take the inputs. We calculate some embedding for them. We then want to retain the position, because of course, the sequence in which the words appear, it matters. And the positional encoding is this weird thing where it encodes using sine waves so that-- it's an orthogonal basis. And so it has nice characteristics.

And then we run it into an attention model that is essentially computing self-attention. So it's saying what-- it's like Word2Vec, except in a more sophisticated way. So it's looking at all the words in the sentence and saying, which words is this word most related to?

And then, in order to complicate it some more, they say, well, we don't want just a single notion of attention. We want multiple notions of attention. So what does that sound like? Well, to me, it sounds a bit like what you see in convolutional neural networks, where often when you're processing an image with a CNN, you're not only applying one filter to the image but you're applying a whole bunch of different filters.

And because you initialize them randomly, you hope that they will converge to things that actually detect different interesting properties of the image. So the same idea here-- that what they're doing is they're starting with a bunch of these attention matrices and saying, we initialize them randomly. They will evolve into something that is most useful for helping us deal with the overall problem.

So then they run this through a series of, I think, in Vaswani's paper, something like six layers that are just replicated. And there are additional things like feeding forward the input signal in order to add it to the output signal of the stage, and then normalizing, and then rerunning it, and then running it through a feed-forward network that also has a bypass that combines the input with the output of the feed-forward network. And then you do this six times, or n times. And that then feeds into the generator. And the generator then uses a very similar architecture to calculate output probabilities, And then it samples from those in order to generate the text.

So this is sort of the contemporary way that one can do translation, using this approach. Obviously, I don't have time to go into all the details of how all this is done. And I'd probably do it wrong anyway. But you can look at the paper, which gives a good explanation. And that blog that I pointed to also has a pointer to another blog post by the same guy that does a pretty good job of explaining the Transformer architecture. It's complicated.

So what you get out of the multi-head attention mechanism is that-- here is one attention machine. And for example, the colors here indicate the degree to which the encoding of the word "it" depends on the other words in the sentence. And you see that it's focused on the animal, which makes sense. Because "it," in fact, is referring to the animal in this sentence.

Here they introduce another encoding. And this one focuses on "was too tired," which is also good. Because "it," again, refers to the thing that was too tired. And of course, by multi-headed, they mean that it's doing this many times. And so you're identifying all kinds of different relationships in the input sentence.

Well, along the same lines is this encoding called ELMo. People seem to like *Sesame Street* characters. So ELMo is based on a bi-directional LSTM. So it's an older technology.

But what it does is, unlike Word2Vec, which built an embedding for each type-- so every time the word "junk" appears, it gets the same embedding. Here what they're saying is, hey, take context seriously. And we're going to calculate a different embedding for each occurrence in context of a token.

And this turns out to be very good. Because it goes part of the way to solving the word-sense disambiguation problem. So this is just an example. If you look at the word "play" in GloVe, which is a slightly more sophisticated variant of the Word2Vec approach, you get playing, game, games, played, players, plays, player, play, football, multiplayer. This all seems to be about games. Because probably, from the literature that they got this from, that's the most common usage of the word "play."

Whereas, using this bi-directional language model, they can separate out something like, "Kieffer, the only junior in the group, was commended for his ability to hit in the clutch, as well as his all-around excellent play." So this is presumably the baseball player. And here is, "They were actors who had been handed fat roles in a successful play." So this is a different meaning of the word play. And so this embedding also has made really important contributions to improving the quality of natural language processing by being able to deal with the fact that single words have multiple meanings not only in English but in other languages.

So after ELMo comes BERT, which is this Bidirectional Encoder Representations from Transformers. So rather than using the LSTM kind of model that ELMo used, these guys say, well, let's hop on the bandwagon, use the Transformer-based architecture. And then they

introduced some interesting tricks.

So one of the problems with Transformers is if you stack them on top of each other there are many paths from any of the inputs to any of the intermediate nodes and the outputs. And so if you're doing self-attention, you're trying to figure out where the output should pay attention to the input, the answer, of course, is like, if you're trying to reconstruct the input, if the input is present in your model, what you will learn is that the corresponding word is the right word for your output. So they have to prevent that from happening. And so the way they do it is by masking off, at each level, some fraction of the words or of the inputs at that level.

So what this is doing is it's a little bit like the skip-gram model in Word2Vec, where it's trying to predict the likelihood of some word, except it doesn't know what a significant fraction of the words are. And so it can't overfit in the way that I was just suggesting. So this turned out to be a good idea. It's more complicated.

Again, for the details, you have to read the paper. I gave both the Transformer paper and the BERT paper as optional readings for today. I meant to give them as required readings, but I didn't do it in time. So they're optional.

But there are a whole bunch of other tricks. So instead of using words, they actually used word pieces. So think about syllables and don't becomes do and apostrophe t, and so on. And then they discovered that about 15% of the tokens to be masked seems to work better than other percentages. So those are the hidden tokens that prevent overfitting.

And then they do some other weird stuff. Like, instead of masking a token, they will inject random other words from the vocabulary into its place, again, to prevent overfitting. And then they look at different tasks like, can I predict the next sentence in a corpus? So I read a sentence. And the translation is not into another language. But it's predicting what the next sentence is going to be.

So they trained it on 800 million words from something called the Books corpus and about 2 and 1/2 million-word Wikipedia corpus. And what they found was that there is an enormous improvement on a lot of classical tasks. So this is a listing of some of the standard tasks for natural language processing, mostly not in the medical world but in the general NLP domain.

And you see that you get things like an improvement from 80%. Or even the GPT model that I'll talk about in a minute is at 82%. They're up to about 86%. So a 4% improvement in this

domain is really huge.

I mean, very often people publish papers showing a 1% improvement. And if their corpus is big enough, then it's statistically significant, and therefore publishable. But it's not significant in the ordinary meaning of the term significant, if you're doing 1% better. But doing 4% better is pretty good.

Here we're going from like 66% to 72% from the earlier state of the art-- 82 to 91; 93 to 94; 35 to 60 in the CoLA task corpus of linguistic acceptability. So this is asking, I think, Mechanical Turk people, for generated sentences, is this sentence a valid sentence of English? And so it's an interesting benchmark.

So it's producing really significant improvements all over the place. They trained two models of it. The base model is the smaller one. The large model is just trained on larger data sets. Enormous amount of computation in doing this training-- so I've forgotten, it took them like a month on some gigantic cluster of GPU machines. And so it's daunting, because you can't just crank this up on your laptop and expect it to finish in your lifetime.

The last thing I want to tell you about is this GPT-2. So this is from the OpenAI Institute, which is one of these philanthropically funded-- I think, this one, by Elon Musk-- research institute to advance AI. And what they said is, well, this is all cool, but-- so they were not using BERT. They were using the Transformer architecture but without the same training style as BERT. And they said, the secret is going to be that we're going to apply this not only to one problem but to a whole bunch of problems. So it's a multi-task learning approach that says, we're going to build a better model by trying to solve a bunch of different tasks simultaneously.

And so they built enormous models. By the way, the task itself is given as a sequence of tokens. So for example, they might have a task that says translate to French, English text, French text. Or answer the question, document, question, answer. And so the system not only learns how to do whatever it's supposed to do. But it even learns something about the tasks that it's being asked to work on by encoding these and using them as part of its model.

So they built four different models. Take a look at the bottom one. 1.5 billion parameters-- this is a large model. This is a very large model.

And so it's a byte-level model. So they just said forget words, because we're trying to do this multilingually. And so for Chinese, you want characters. And for English, you might as well take

characters also. And the system will, in its 1.5 billion parameters, learn all about the sequences of characters that make up words. And it'll be cool.

And so then they look at a whole bunch of different challenges. And what you see is that the state of the art before they did this on, for example, the Lambada data set was that the perplexity of its predictions was a hundred. And with this large model, the perplexity of its predictions is about nine.

So that means that it's reduced the uncertainty of what to predict next ridiculously much-- I mean, by more than an order of magnitude. And you get similar gains, accuracy going from 59% to 63% accuracy on a-- this is the children's something-or-other challenge-- from 85% to 93%-- so dramatic improvements almost across the board, except for this particular data set, where they did not do well.

And what really blew me away is here's an application of this 1.5 billion-word model that they built. So they said, OK, I give you a prompt, like the opening paragraph of a *Wall Street Journal* article or a Wikipedia article. And you complete the article by using that generator idea that I showed you before, that just uses the language model and picks the most likely word to come next and emits that as the next word.

So here is a prompt that says, "A train carriage containing controlled nuclear materials was stolen in Cincinnati today. Its whereabouts are unknown." By the way, this is made up. I mean, this is not a real news article.

And the system comes back with a completion that says, "The incident occurred on the downtown train line, which runs from Covington and Ashland stations. In an email to Ohio news outlets, the US Department of Energy said it's working with the Federal Railroad Administration to find the thief," et cetera. This looks astoundingly good.

Now, the paper from which this comes-- this is actually from a blog, but they've also published a paper about it-- claims that these examples are not even cherry-picked. If you go to that page and pick sample 1, 2, 3, 4, 5, 6, et cetera, you get different examples that they claim are not cherry-picked. And every one of them is really good. I mean, you could imagine this being an actual article about this actual event. So somehow or other, in this enormous model, and with this Transformer technology, and with the multi-task training that they've done, they have managed to capture so much of the regularity of the English language that they can generate these fake news articles based on a prompt and make them look unbelievably realistic.

Now, interestingly, they have chosen not to release that trained model. Because they're worried that people will, in fact, do this, and that they will generate fake news articles all the time. They've released a much smaller model that is not nearly as good in terms of its realism.

So that's the state of the art in language modeling at the moment. And as I say, the general domain is ahead of the medical domain. But you can bet that there are tons of people who are sitting around looking at exactly these results and saying, well, we ought to be able to take advantage of this to build much better language models for the medical domain and to exploit them in order to do phenotyping, in order to do entity recognition, in order to do inference, in order to do question answering, in order to do any of these kinds of topics.

And I was talking to Patrick Winston, who is one of the good old-fashioned AI people, as he characterizes himself. And the thing that's a little troublesome about this is that this technology has virtually nothing to do with anything that we understand about language or about inference or about question answering or about anything. And so one is left with this queasy feeling that, here is a wonderful engineering solution to a whole set of problems, but it's unclear how it relates to the original goal of artificial intelligence, which is to understand something about human intelligence by simulating it in a computer.

Maybe our BCS friends will discover that there are, in fact, transformer mechanisms deeply buried in our brain. But I would be surprised if that turned out to be exactly the case. But perhaps there is something like that going on. And so this leaves an interesting scientific conundrum of, exactly what have we learned from this type of very, very successful model building?

OK. Thank you.

[APPLAUSE]