Word meanings

Professor Eric Atwell: Hello, this is Eric Atwell, and this lecture I'm going to talk about word embeddings or meanings and vector semantics. For more details, see chapter 6 of Jurafsky and Martin's textbook Speech and Language Processing.

Now, representation of word meanings is core to natural language processing and tech analytics. It's the theoretical underpinnings of a lot of research and methods use nowadays.

I have to say also, this is going to be quite a long lecture because there's a lot of detail. Even so, I'm not going to cover everything in the chapter, so I recommend you read the chapter for more detail.

And if you look at Wikipedia, it explains an embedding. A natural language processing, a word embedding, is a term used to represent the word meaning for text analysis.

Typically, a real valued vector that encodes the meaning of a word such that the uses are closer in vector space are similar in meaning. We'll see a better explanation of that shortly.

So I want to look at what does it mean for word-- what is word meaning? Then look at some of the background on vector semantics, and how you can represent words in terms of vectors, and then how you measure similarity between two vectors, which is the same as word similarity, in effect, using the cosine function.

Then, look at some ways of making these vectors a bit more realistic, better, in terms of using TFIDF, or term frequency divided by document frequency, or pointwise mutual information as an alternative to raw frequency.

And we're going to look briefly at word2vec, a deep learning tool, which allowed you to use deep learning to learn word embeddings from large amounts of text, and then finally, some of the interesting properties of these sorts of embeddings in terms of doing understanding of text.

So the first issue is what do we mean by word meaning? What do words actually mean? So we've seen so far looking at n-gram models or text classification models, you essentially take a word as a string of characters or an index to a vocabulary list.

And that's not very satisfactory. It means that class and classes, because they're two different strings, mean different things. Or wedding and marriage, which have different strings, therefore they have nothing in common in terms of meaning.

In many other applications of AI, for example, in the non-representation module or in the machine learning and deep learning module where you look at computer vision applications typically you might say, the meaning of a picture of a dog as opposed to a picture of a cat, one is labeled D-O-G and the other is labeled C-A-T.

Or in philosophical logic you might say, for all x, if x is a dog, then x is a mammal, which applies that D-O-G is the meaning of dog and M-A-M-M-A-L is the meaning of mammal. But that doesn't really tell you very much, just saying you're writing in capital letters.

In linguistics, this has been a joke for some time. Back in the 1960s, famous linguist called Barbara Partee asked what is the meaning of life? And meaning of life is as L-I-F-E. Well, that's not particularly funny, and furthermore, it's not particularly meaningful.

In computer vision classification it may make sense to say an image, which is some large number of pixels, the label of the image is something like dog, or cat, or if you're doing person recognition, it's the name of the person. But that is a character string.

And saying the meaning is a character string may work for images. But if the actual instance or thing you're trying to represent is a character string, then saying a character string-- its meaning is a character string doesn't help very much.

So what should a meaning be? Well, first of all, you can look at what sorts of things would a theory of a word meaning help with? Is a desiderate? Desiderate is a Latin word meaning "things that are "desirable.

And in linguistics, the study of lexical semantics-- lexical means to do with words-- has been going on for some time. And they've looked at what it would be useful to have.

Well, one thing is that you might say, look at a dictionary. If you're learning what a word means, then look it up in a dictionary. And for example, the entry or lemma-- lemma means a dictionary entry-- for mouse would have the spelling of the word mouse, M-O-U-S-E as a character string.

It might say it's a noun. N means noun. And then it would have a number of senses. Sense one is any of numerous small rodents. Sense two is a hand-operated device that controls a cursor. That's the modern sense of mouse.

So a sense or a concept is the meaning component of a word. And lemmas can have more than one sense. They can be Polysemous. And you might think, OK, it ticks some definitions.

But you're still defining the meaning of sense one as a sequence of words or a sequence of character sequences. Is a sequence of words any better than just a word?

Saying a mouse means mouse is not very sensible. Saying mouse means any of numerous small rodents-- you've got a number of words, but is that actually useful? We're not so sure.

Another thing you want to have, is you want to be able to somehow capture the fact that two words can mean the same thing. Synonyms have the same meaning in some or all contexts. So couch means the same as sofa, big and large, automobile and car, and so on, even though they are different character strings.

Not all contexts, so words which are in the dictionary as synonyms may be actually slightly different in terms of politeness, or how slang they are, or in a particular genre or topic they mean different things. Like mouse in a computer text means something different from mouse in a biology text.

A simple example, water and H2O, while you wouldn't come across H2O in a surfing guide, for example. You would use the word water in that sense.

Or even big and large, fairly common words which seem to mean the same thing, but my big sister means my older sister. Whereas my large sister means my sister who wears large size clothing.

And they're not necessarily the same thing. I could have a rather petite older sister and a rather larger younger sister.

Another thing you want to have in meanings, is you'd expect, in general, difference in form leads to a difference in meaning. So two words which are like small and large should mean different things. And this has been known for quite a long time.

So here's an extract or a quote from Abby Gabriel Girard from 1718, which is like 300 years ago. [SPEAKING FRENCH] "I do not believe there is a synonymous word in any language."

There are no exact synonyms. However, you can have similarity, words which are similar in meaning. They're not synonyms, but share some element of meaning. Like car and bicycle are not synonyms, but they both have wheels. They're both methods of transport.

Or cow and horse, they're both animals that you might find on a farm, even though they clearly are different. So synonymy, synonym means they are exactly the same. And you'd think words are either synonyms or they're not synonyms. It's a nice binary relationship, one or zero.

Whereas, similar is on a scale. And psychologists like to do experiments with humans. And a typical experiment is just give pairs of words and ask how similar are they on a scale of 1 to 10, let's say.

So vanish and disappear, if you ask several people, they will tend to say they are very similar. And in experiments is a SimLex-999 data set from Hill to 2015. And they tried this out and lots of people.

And on the whole, people would think that vanish and disappear are nearly a 10. Whereas, take two words at random like whole and agreement, they're nearly zero. And there is some measure of similarity. It's a bit hard to think of what it is, but there is some similarity.

So similarity is synonym like. There is also another avenue of similarity, which isn't quite the same thing. Relatedness or association, that is words which are related in some way, even though they're not really synonyms or even close synonyms.

For example, coffee and tea are similar because they're both things you drink. There's coffee and cup, are related more because they're a collocation. Remember, in sketch engine you can look up collocations, words which appear together very often in a corpus.

But that doesn't mean that cup has the same meaning as coffee, but rather not in the same sense. So similarity is in terms of being near synonyms. Relatedness or association is words which come together more than you might say at random.

There are also semantic fields or topics, words which come together in a document on a particular topic. Like in documents to do with hospitals or medicine, you'd expect words like surgeon, scalpel, nurse, hospital.

Even though they are not synonyms, they're related because they belong to the same topic area. And there are some other examples.

Another relation apart from synonym is antonyms. And so I had a PhD student, Lulu al Dubai, who looked at there are data sets containing synonyms, but they don't have antonyms. And she added this for the Arabic word net, which contains synonyms. She added antonym relations.

So these are opposites with respect to only one feature of meaning. Otherwise they are very similar. They have a lot in common. So for example, dark and light are both about intensity of luminosity.

So they're both similar in most terms, except for this one feature. The same for short and long, or rise and fall. So we can start to see that meaning could be broken down into features.

In much the same way as-- in machine learning an instance is characterised by a number of features. And you can have some features being the same or similar, while other features are not the same or similar.

So dark and light, most of the features are more or less the same except for this one intensity feature, which is high in light and low in dark. So more formally, antonyms can define a binary opposition at opposite ends of the scale, like long and short, or fast and slow.

Or they can be what's called reversives. They reverse the duration or the direction of travel. OK, so there's a number of these different features.

Another feature that is relevant and particularly topical because of sentiment analysis-- it seems to be very popular in natural language processing and tech analytics, and that is sentiment. So words can be happy or sad, have effective meanings. Positive is happy. Negative is sad.

It can also have rather more subtle connotations. So happy and sad is an obvious opposition. But you can also have connotation. So copy and fake mean the same thing, more or less, except copy is positive, and fake is negative.

Or replica a knockoff, replica is a positive thing, whereas a knockoff is a negative thing. Or reproduction sounds good, whereas forgery sounds bad. And in general, there are lots of positive sentiment terms and negative sentiment terms like great versus terrible.

So we can see that word meanings can vary on a number of different dimensions. Even in terms of connotation or sentiment, there are at least three dimensions. There is the pleasantness of the stimulus. So love is pleasant, whereas toxic is unpleasant.

Also in terms of the intensity of emotion, so elated or frenzy is very intense, whereas mellow or napping is very not intense. And you also have the dominance, the degree of control. So powerful and leadership are controlling, whereas weak or empty are not controlling.

So for some sentiment lexicons it's not just positive or negative, but there are several different features or dimensions which you try to measure.

So far we see that meaning is not straightforward, but we start to get this idea that there are a number of different features or dimensions you can measure. Maybe you can measure these separately.

So first of all, word senses rather than word meanings, so a word can have several different senses. And for each separate sense you have different relationships.

And then each word sense can have relations to other word senses in terms of synonyms, or possibly antonyms, or if they're not exact synonyms, they can be similar. Or they can be related in that they come together a lot. Or they can have common connotations. They can both be positive or both be negative.

OK that's some linguistic theory about word meanings. Now, how do we get on to capturing these in vectors? And this is very important for this module. Because as it says in Jurafsky and Martin, every modern NLP algorithm uses embeddings as a representation of word meanings.
So if we want to represent the meaning of a document, then you use an embedding to represent the words. And then you can see how close two documents are by using how close the set or sum of embeddings are. So these vectors are very important.

So how can we build a theory of how to represent word meanings that accounts for at least some of these desiderate? Well, let's look at vector semantics. This is the standard model in language processing. And it manages to handle lots and lots of our goals.

OK, so in terms of philosophy and linguistics, people had this idea-- for example, Wittgenstein, a famous philosopher, said, "the meaning of a word is its use in the language." So if you want to capture the meaning, then you capture lots of examples of the word's use. And those examples can somehow give you the meaning.

Or John Firth, a linguist, J R Firth, said, "a word is characterised by the company it keeps." Now, if it's the collocations, that's a bit more precise than just its use. If you look at the collocations, the words immediately before and after, and you look for 100 copies of a word, and you look at the hundred words that appear before and 100 words appear after, that represents its meaning.

And then you start to get a vector of 100 words before and another 100 words after, maybe 200 items in the vector. And that represents the meaning of the word.

So usage is the words around it. It's environments of the word. So Zelig Harris, another linguist, said, "if A and B have almost identical environments, we say that A and B are synonyms."

So if you find two words like little, and look at the words before and after little in 100 cases, and another word like small, and the words before and after small in 100 cases, if those set of environments are very similar, then probably little and small are synonyms.

And we can do this for unknown words. So the whole point of doing it computationally is the computer doesn't know the meaning of little, or little of small, so it can't really work out that they are similar.

But let's say, for example, in English-- this is an American borrowing more than a British borrowing, but it's in the textbook written by Americans. Ong choy, what does that mean?

Well, if you go into the British National corpus or rather maybe the American National Corpus, or N 10 10 in sketch engine, you can search for ong choy. And you'd find sentences like, ong choy is delicious sauteed with garlic, or ong choy is superb over rice, or ong choy leaves with salty sauces.

And then you can look for other words which appear in those sorts of contexts. And you may find spinach sauteed with garlic over rice, or chard stems and leaves are delicious, or collard greens and other salty, leafy greens.

So those sorts of connotations let you to conclude that ong choy is some sort of leafy green a bit like spinach, chard, or collard greens, assuming you know what spinach and so on are. And that's the idea.

Another thing of course you can do is you could Google it and find it ong choy-- you can find a picture of ong choy. So in computer vision terms, you say this picture, the meaning of his picture is ong choy because the label is there. Or maybe it be a Chinese label, which means the same thing.

OK so we can define in text the meaning by its linguistic distribution. The meaning of word is its distribution in language use, meaning its neighboring words or its grammatical environment. The grammar means the words before and after in this case.

Furthermore, you might say, if you've got a vector, then you may want the vector-- if, let's say, 100 items in the vector, that's equivalent to saying a vector is a point in 100 dimensional space. And then you can see if two words are similar by seeing how far they are apart in this 100 dimensional space.

To make it a simpler because it's hard to deal with 100 dimensions, let's look at three dimensions. We already said that in terms of effectiveness or the sentiment value of a word there's pleasantness or valence, there's arousal or intensity, and there's dominance or control.

And a word like love has a score of 1 in terms of pleasantness, but not so high for arousal, and not particularly high for dominance. Each of these words has a point in this three dimensional space. So the connotation of a word, at least, is a vector of three values, and therefore it has a point in three dimensional space.

OK, so the vector semantics means defining a meaning by linguistic distribution and using linguistic distribution to populate the vector, to give the numbers in the vector. And that then gives you meaning as a point in multidimensional space.

So each word is a vector. It's not just the meaning of good is good, or the meaning of good is word number 45 in the dictionary. That doesn't help us very much.

And furthermore, the meaning of word is a point in space such that similar words, words with similar meanings, should be near to each other in this semantic space. And we can build this space automatically by counting up the words which are nearby in a large sample of text.

So, here-- oops, sorry I went too fast.

Here's an example, if we take a large sample of text, such as N 10 10, you find that not good, and dislike, and bad, and worst are in one part of space. Whereas good and fantastic are in a different part of space.

And there's also, in terms of sentiment, another part of space for words like to, and that, and now, which don't have any particular sentiment attached to them. So it's not really binary. It's sort of 1 plus minus and 0 in terms of sentiment. And if you can map these things, then you should do it.

Of course, this is assuming you have 100 dimensional space, which you can then using mathematics squashed down or map onto two dimensional space, so that we can display it in a slide. So all of this works on the assumption that you can do some sort of embedding.

Embedding means taking a large dimensional space and mapping it out into a small dimensional space. And that's mathematics. And there are algorithms for doing that, which I'm not going to go into. But it is typically possible.

So we define the meaning of a word as a vector. And if you look at Wikipedia for embedding, it will tell you mathematical embedding from a space with many dimensions per word to a continuous vector space with a much lower dimension.

So for example, remember the LOB corpus, Lancaster/Oslo/Bergen corpus, that was one of my first jobs was to work on collecting this one million words of British English. It contained about 50,000 word types. There's one million word tokens. That means there's 50,000 different words.

So if you want to look at any particular word, then the set of context it could have-- it could have up to 50,000 words before or after, 50,000 other words before or after.

In other words, you'd have to represent as a vector the meaning of a word is a vector of 50,000 possible context words. But that's far too big for doing any processing. I tried it back in 1986, and our computers just weren't up to it at that stage.

So we had to deal with a much smaller vector of the 100 most frequent words only. And that was just about manageable on 1986 computers. Or the University mainframe at Lancaster University, I ran it for about a month and it managed to do something.

And if you read my 1986 paper, you can actually see the output. It's not very impressive at all. But that gives you some idea of how difficult it is to deal with very large numbers. Nowadays we can do it.

And we'll see word2vec can take 50,000 vectors and embed them into 100 or maybe 1,000 values. And a vector of a hundred or even a vector of 1,000 is much more manageable. This is the standard way to represent meanings in natural language processing.

Every modern NLP algorithm uses embeddings to represent word meanings, so you need to know this. It also gives us a much more fine grained model of meaning to measure similarity.

If you can take an embedding of 1,000 numbers, or even a hundred numbers to represent the meaning of wedding, and then 100 numbers to represent the meaning of marriage, then assuming that wedding and marriage appear in the same sorts of contexts, you should be able to find that the vector for a wedding is quite close to the vector for marriage in this meaning space.

So why vectors? Well, in sentiment analysis you might want to say that the previous word was terrible. And it doesn't require exactly the same word to have terrible to appear before or after.

If you're doing sentiment analysis on just words, then you have to have a whole lot of features which represent bad words, and another whole lot of features said good words. And each of these is a separate feature.

Whereas, if you have embeddings, then the word vector could be a set of numbers. And in the test set we have a vector, which is not exactly the same vector, but is similar.

And that lets us say that actually, this is probably a bad sentence because it contains a word which isn't the same as terrible, but has a very similar vector. So we can generalise to similar but unseen words. You could have words in the test set, which you haven't seen in your training set.

But the vectors are similar in meaning, therefore they do count as reasonable training tests test items. Whereas, before, if you had a word in your test set, and it wasn't in the training set, it was just out of vocabulary.

You had to do something like add one smoothing or give it an unknown value, which wasn't very helpful. So unknown doesn't tell you anything about the sentiment, whereas a similar vector could help.

OK, we're going to look at more complicated vectors, presently, including TFIDF. That's term frequency inverse document frequency, or inverse means divided by, term frequency divided by document frequency.

That's commonly used in information retrieval. That's like Google search or library search. We're also going to look at very sparse vectors as opposed to very, very dense vectors.

So basically, to build the vectors, the simplest thing you can do is count up the nearby words. And then the counts are for nearby words give you some idea of what the meaning of the word is.

On the other hand, word2vec, this new deep learning model, it basically took very sparse vectors and mapped them onto very smaller but denser vectors. And these are much better, representations created by training a classifier to predict whether a word is likely to appear nearby.

And so we'll see these two different examples. Both of them are vector representations of meaning, but they have different pluses and minuses.

So from now on, we're going to do computing with meaning representations rather than a string representations. And Jurafsky and Martin quote this neat poem. "Net's off a fish, once you get the fish, you can forget the net." Similarly, words off a meaning, and once you get the meaning, you can forget the words.

So this is another nice thing. If you had these embedding representations for Chinese words and embedding representations for English words, then if you're really lucky, the embedding representation for a particular concept is the same for the Chinese or for the English.

And you don't need the individual words. Or maybe you can use this to help you in translating. More on that later, that's enough for vector semantics.

OK, now we're going to look at some more about the interaction of words and vectors. Let's look at some actual examples. In information retrieval, that's finding the right document or finding the right web page, which matches your keywords, this is a common sort of model.

You might think of a term document matrix. So a document could be a web page, or a verse from the Quran, or in this case, it's one of the plays of William Shakespeare.

So as a British academic, I have to recommend that you read or even watch some of the plays by William Shakespeare, because they're supposedly very good. Anyway, it's a nice example, because all of these plays are, all the texts, are available for you are freely online.

They're no longer a copyright. So the total works of William Shakespeare is about nearly a million words all together. And you can use this as an example. A lot of these are plays. So here's four plays, As you Like It, Twelfth Night, Julius Caesar, and Henry V. V is Latin number five.

And here's four example words and the distributions. So we can see a matrix of four documents and for four words. Battle occurs once in As You Like It, 0 times in Twelfth Night, 7 in Julius Caesar, and 13 times in Henry V, and so on.

So each document could be represented by a vector of words. So As You Like It, although it is a play, in frequency terms we just say it consists of 1 battle, 14 good, 114 goods, 36 fool, and 20 wit.

In fact, you don't even have to know the words. You just say, the meaning of As You Like It is 1, 114, 36, 20. So that's a meaning of a document. And you can visualise spaces in the document vector space.

So for example, if you take the frequency of battle as against the frequency of fool, we'll see that each of-- just in terms of those two words, each document has a space in two dimensional space. So Henry V has 4 fools and 13 battles.

And Julius Caesar has 1 fool and 7 battles, whereas As You Like It has 36 fools and only 1 battle. And Twelfth Night is even more extreme. It has 58 fools and 0 battles. So you can start to see that there is a, it's good at dividing up. You can see similar plays.

It looks like Julius Caesar and Henry V are similar because they are similar vectors. And As You Like It in Twelfth Night are two comedies. And they are similar because they both have lots of fools and wit.

And the comedies are different from the other two, which are more about famous characters in history who did a lot of fighting. So comedies have more falls and wit and fewer battles. But you can also do the same sort of thing for word meanings.

So if you do the vectors the other way, then the meaning of battle is 1, 0, 7, 13. And the meaning of good is one 114, 80, 62, 89. So battle is the kind of word that occurs mainly in Julius Caesar and Henry V. While fool is the kind of word that appears in comedies as opposed to the historical characters.

So that's another way. But wit is just 20, 15, 2, 3. And you can see that wit and fool are similar in meaning, and battle is quite different from those. The two words are similar meaning if their context vectors are similar.

And here's another example, if you take a much larger corpus, then you can count up for every word how many times it appears to other words. So here's an example. The word aardvark is at beginning and zoo is at the end. These are in alphabetical order.

And so for example, for the word cherry, if you look at the words in its context, aardvark is never close to cherry, computer a couple of times, data few more times, and so on. And pie is very common next to cherry.

Whereas digital occurs much more commonly next to computer and data and hardly ever next to pie and sugar. So digital and cherry mean very different things.

Whereas digital and information, if you just take the two features computer and data, you can see that digital and information are very similar because they both have high numbers for computer and for data.

It turns out that information is much more frequent than digital. Therefore they're the actual frequencies for computer and information being in the context are much higher. But the digital still means the same sort of thing as information. OK, so that's an idea of words and vectors.

Now if we want to measure similarity, then as we saw in that previous diagram, what matters is that in the vector space they are very close to each other. And we can measure-- if we map this down into two dimensions, then you can measure this angle.

And this angle, if the angle is small, then they're very close to each other. If the angle is large, then they're far apart. And cosine is a metric of how small or large this vector is, this angle is, sorry. So we compute the word similarity using dot product and cosine similarity.

The dot product between two vectors-- and for further explanation, see that the textbook. But basically for a dot product or for vector 1 dot vector 2, then you take for each-- the first value in vector 1 and the first value in vector 2.

If v is a vector and w is vector, then you take a v1 w1, and add on v2 w2, and on v3 w3, and so on. So you sum together, for each position, that the values at vector v and vector w, and that sum gives you the dot product.

The dot product tends to be high when two vectors have large values in the same dimensions. So going back to information that has computer as a high value and it also has data computes as a high value.

So a dot product gives us a useful similarity match-up between them. However, as a rule, dot product is a bit limited. Because if the vector is longer, in other words, if there's lots of high values, then it will have a higher value overall.

So what you really want to do is take into account the length of the vector. So rather than just the length, the square root of the sum of the squares of the values. So take into account all the values, square them all, add them together, and take the square root.

That's a way of if you're normalising for the frequency of the word. So a very frequent word like of, and the, and you will be very frequent, and will co-occur with lots of words. So in other words, the vectors, all the words which co-occur with of, there will be a lot of them. Therefore we'll have a very high dot product value.

So you want to normalise or divide by the frequency. And this is-- so what you actually do is you take the dot product and divide it by this metric for vector 1 or v and for vector 2 or w. And that gives us this equation.

And if you like maths, then see the textbook for further explanation of this. If you like maths you probably immediately see what this means. If you don't see what this means immediately, then take it that it's basically a way of normalising the dot product for vector length, by dividing the product by the lengths of the two vectors.

And this actually turns out to be the same as the cosine of the angle between the two vectors. Because the dot product is-- well by taking a dot product divided by the length of A length of B gives you the cosine of the angle.

Cosine is a nice metric very widely used in engineering. So it's a value that ranges between 1 and minus 1. So if the vectors are positive or approaching one, that means the words are very similar. But actually minus 1 means that they're in the opposite direction.

So they're similar, but opposite in some feature. And that can be problematic. So actually what you tend to do is to say, that all the raw frequencies are non-negative. We don't have any negative frequencies. Therefore we only have positive values of this cosine metric.

You only take the coastline in the range 0 to 1, not in the range of minus 1 to 1. So here's a simple worked example, taking just cherry, digital, information, and the contexts around this being pie, data, or computer.

Cherry occurs very frequently with pie, but not very often with data or computer. Whereas information occurs not very often with pie, but a lot of the time with data and computer.

So if you work out the cosine metric for cherry and information, then it's 442 times 5 plus 8 times-- this is for information-- 8 times 3,982 plus 2 times 3,325-- we see what--

We're cross multiplying the first vector value, and the second vector value, and the third vector value for cherry and for information. And then divide that by this normalising feature. For cherry, the normalising is 442 squared plus 8 squared plus 2 squared, and the square root of that.

And for information, the normalising value, which takes in the length of the information vector, that's 5 squared plus 3,900 squared plus 3,300 squared and the square root of that. And the normalised value overall is 0.017.

Now, I've gone through this rather quickly. If you feel uncomfortable, then work it out yourself with pen and paper just to check that I've done the maths right. Actually, I'm cheating. I'm just taking the maths straight from the Jurafsky and Martin textbook.

So I'm trusting that Dan Jurafsky got it right. And whether or not he got it right strictly doesn't matter. This is just for illustrations. The point is that the cosine metric measuring-- the distance between cherry and information is quite small. It's quite close to zero.

Whereas the equivalent mathematics-- do the calculation for digital and information. You get a very high value close to 1. Notice also if you remember, digital is much less frequent than information. Therefore all the values for information are high.

But what matters is the angle between digital information in that two dimensional diagram is very small. In other words, digital and information, the points in space can actually be quite far apart. But what matters is the angle between them is very small. And therefore, the cosine is nearly one.

If the angle was 0, then the cosine would be 1. If the angle was 90 degrees, then the cosine would be 0. In other words, at 90 degrees they have nothing in common at all.

So if we map this in two dimensional space, you can see-- if dimension one is pie and dimension two is computer, then cherry and digital is nearly 90 degrees, and therefore they have virtually nothing in common. Whereas digital and information is nearly zero, and therefore that angle is nearly zero, which is one.

OK, so that's cosine as a metric for computing word similarity. We've looked at frequencies, and raw frequencies do have some problems. Raw frequencies tend to over empower words which are very frequent, and not deal very well with lower frequency words.

And also not deal very well with comparing low frequency with high frequency words. Frequency is clearly useful. If sugar appears a lot near apricot, that is useful. The frequency of sugar next to apricot is good.

But words like the can also appear next apricot a lot. And we want to know that sugar is closer in meaning to apricot, whereas the is not very important in meaning.

Some of the very early experiments in embedding are referring to a paper I wrote back in 1986, just use raw frequencies. And because the only thing we have were raw frequencies, it could only deal with the most frequent words.

Again, partly because you couldn't deal with thousands of words at those times. The University mainframe computer running for a month could only cope with the most frequent words, and only cope with a very small corpus.

And so it could only deal with words like the, and it, and of. And it notice that, for example, of and in were similar in meaning in the sense that it's sort of linguistically they were both prepositions. Whereas the is a bit more different because it's not a preposition. But that's not very helpful for real content words.

So how do we balance this? How do we deal with this? Well, one way is something called TFIDF, which is term frequency multiplied by inverse document frequency.

And that works because words like the and it have a very low inverse document frequency, and therefore they are normalised out. And words like sugar have a higher inverse document frequency, and therefore they get a higher weight.

Another way of doing this is point-wise mutual information. If we take the likelihood words like good appear more often with great than you would expect by chance, whereas the words like the appear all over the place, and therefore by chance they are very likely to appear.

So if you divide by-- if we take a pair of words and how frequently they occur together, like the and good, maybe quite as likely as great and good. Whereas we take the probability of the individual words and divide by that, then the word like the has a very high likelihood.

And therefore if we divide by it, then downgrades the effect of the. We look at this in some more detail. Well, first of all, let's look at TFIDF.

So we can count up how many times a particular word term, like cherry, appears in a number of documents. If you have a corpus, and the corpus is made up of documents, then we count up how many times the word appears in all of the documents.

But remembering back from Markov modeling or n-gram modeling, you actually have to add one or do something to normalise to deal with 0 counts. So we don't want a count to be zero. So if we add one, that's normalising or smoothing.

And furthermore, if we take probabilities, then the problem with probabilities is they're very small. They're numbers in the range of 0 to 1 or fractions. And if we multiply together several fractions, you get vanishingly small fractions.

So it's better to take logarithms. So the term frequency is essentially the logarithm of the count plus one, rather than just the actual count. The a document frequency is the number of documents the term appears in. And this is not the frequency in the total document collection.

So Romeo, for example, in Shakespeare appears 113 times in the whole of the Shakespeare corpus, but actually only appears in one document, one play.

Action, on the other hand, is also quite frequent. It appears 113 times in the corpus of Shakespeare. But it occurs in 31 different documents. Therefore Romeo is more distinctive or meaningful than action.

And we see this in the inverse document frequency. That is how many documents it appears in out of that total set of documents. And again, we take the logarithm rather than the actual frequency. So the document frequency of Romeo is one. The inverse document frequency using this logarithmic function is 1.57.

Whereas, another word like salad appears in two documents, and therefore has an inverse document frequency of 1.27. Or Falstaff appears in four plays, and has an even lower inverse document frequency and so on.

You can see that words like good and sweet, which appear in all of the documents, all of the plays, have a very low inverse document frequency. That's in terms of Shakespeare. In general, the document is whatever it is we're counting. So whatever is the unit in terms of documentation.

So if you're dealing with Wikipedia, then each Wikipedia article counts as a document. And the corpus is the set of all Wikipedia articles. Or in Quran AI research, Leeds University does a lot of research on Quran and understanding the Quran, so therefore each verse could be a document.

So the TFIDF weight gives a value for a word where it's the term frequency, how many times the term appears in the overall corpus multiplied by the inverse document frequency, or divided by the document frequency.

Let's look at an actual example, going back to Shakespeare. I remember this. The TFIDF weights for things like battle doesn't appear in Twelfth Night at all, so it gets a zero inverse document frequency.

Words like good are highly frequent, but good appears in lots and lots of, in fact, good appears in all of the plays. Therefore, although good is much more frequent in As You Like It and less frequent in the other ones, it actually gets a zero score overall, because it adds no information at all in terms of being specific to a particular document.

So good actually gets a zero score, and we just ignore it in TFIDF calculations. OK, so let's TFIDF. OK, now let's look at--

That's one way of doing it. The other way of normalising frequency is this PMI, or pointwise mutual information. Pointwise just means the overall model depends on individual words, individual points in the space. What's the mutual information?

Do events x and y co-occur more often than if they were independent? So the mutually is how often they occur together as opposed to independently.

So the pointwise mutual information between two words is how many times do the words occur together more often than if they were independent. So we can take the probability of word 1 and word 2, divide it by the independent probability of word 1 and the independent probability of word 2.

Only, remembering that we work in logarithm space rather than in actual probability space, so that we don't have underflow. We don't have very tiny numbers. However, this straightforward pointwise mutual information can be negative, because the logarithm can be negative.

And we don't really like negative values. So if things are less than we expect by chance, we simply take 0 rather than a negative number. This is because you have to have enormous corpora to be able to calculate these probabilities realistically.

Imagine we have two words, W1 and W2, and the probability of each is one in a million, then to calculate the probability of W1 and W2 occurring together, we'd really have to have a-- so the probability of 1 and a million times 1 in a million is one in 1,000, 1,000 million or a billion, depending on whether you're British or American, or 10 to the minus 12.

So we really don't have corpora that size. So we can't really measure unrelatedness. We can't really say that they are less than expected by chance.

So basically, we take the positive PMI, or PPMI, between two words is the probability that they occur together divided by the probability of the first word times the probability of the second word. And you take a logarithm of that.

And if a logarithm is positive, then that's the value. But if the logarithm is negative, then we take 0. PPMI is basically a number between 0 and very large.

And this is done, again, on this term context matrix. And going back to cherry, strawberry, digital, information, and so on, computer, data, result, pie, and so on. We can see that cherry doesn't come with computer very often.

Digital does come with a computer an awful lot. And we want to take out the probability of the words appearing together divided by them being independent. And if this probability turns out to be, or the logarithm turns out to be negative, because the probability is very small, then we take it as zero.

And here's some examples. Let's say information and data, if we want-- intuitively, information and data are related. So you want this number to be better than 0 at least. And we take the probability of information given--

Let's say the information given data is 3,982, and there are altogether 111,716 words all together. So the probability of information out of all words is 0.3399 apparently.

The probability of just information, information occurs 7,000 times altogether out of 111-- Sorry, that should be 11,716. I think there's a typo there. OK, it's 11,716 is the total number of words. And that has a probability of 0.65. So information, out of all these words, has that probability.

The count of data, whereas data is 5673 occurrences of data out of 11,716. That's 0.4842. And if we put all these figures into that, then we have with PPMI matrix. So we do these calculations for each of these words.

If you're not so sure, look at the textbook and do the calculations yourself. But the point is you end up with a matrix of scores, and these scores are either 0 or slightly higher than 0. And some of these scores are very high, so that the PPMI of cherry pie is 4.38.

And it gives you a likelihood, or a score, for cherry and pie co-occurring. And if digital and pie are very unlikely, then you get a 0 score. And one nice thing about this is that you get 0 scores for lots of things that just don't occur together.

You only get non-zero scores for things which should actually come together, which are meaningful in some sense. So cherry, pie, cherry, sugar, strawberry, pie, strawberry, sugar, they have non-zero values. Whereas everything else gets a zero value.

Digital and computer come together. Digital and data come together. It turns out the digital and result are not very likely, whereas information and result are quite likely. OK, but in general you only get non-negatives for things which should actually occur together, remembering that you have a logarithm of all these things.

OK so PMI is biased towards infrequent events. So very, very quite rare words can have very high PMI values. That is a problem. And solution is to do some add one smoothing just as we did in n-grams.

For very infrequent words, you don't take the probabilities, but you add 1 to the counts to give you smooth probabilities. And that then stops biasing towards infrequent events.

In the textbook there's more about this, but for now, let's just say that PMI is one way of dealing with probabilities, and TFIDF is another way of dealing with probabilities. And both of these will give you vectors as a way of representing the meanings of words.

And the vectors will have zero values in them for things which just don't go together. So you end up with a very large vector, but most of the values are 0. And you only have to deal with the non-zero values.

And that gives you quite sparse vectors. TFIDF or PMI are ways of giving you meaning vectors which are long. Typically, you have a vector which has a value for every word in the vocabulary.

So for example, for the LOB corpus, there are 50,000 different words in it. Or for the works of Shakespeare, there are 20,000 different words in that Shakespeare's used. So the vocabulary size can be 20,000, 50,000, or even bigger if you're dealing with the web or some other larger corpus.

And the meaning of any word is a vector of-- the meaning of any word in Shakespeare is a vector of 20,000 values, or the meaning of any word in the LOB corpus is a vector of 50,000 values. But they are sparse in that almost all of the elements are 0.

Doing their calculation by TFIIDF, the inverse document frequency is often zero or one. Therefore, whether or not you add one, you end up with most of the elements being 0.

It would be nice if there was a way of having vectors which are much shorter, maybe 1,000 or even down to 50 values, and all of the values are non-zero. So we don't want we want all the values to be in the range, say, 0 to 1.

And because the calculations then get much easier. You want to compare two words. Is wedding similar to marriage? Well, you'd like to compare to 1,000 value vectors rather than to 50,000 value vectors, hopefully.

So that you want short, dense vectors. Well how can you do this? Well, it turns out that the sort of methods that are used back in 1986 only give you long sparse vectors. And that was part of the reason it took a month to compute anything on the University mainframe computers at that time.

Now, we have very powerful computers and they can do things called embedding. And embedding, as we saw, is a way of mapping down or squashing down very large vectors into much smaller vectors. Short, dense vectors can be easier to use as features in machine learning.

If you do machine learning, you only want to have a limited number of features. If you have 1,000 features, it's still quite difficult, but it's easier than if you've got 50,000 features. Also, these dense vectors can maybe generalise better than the explicit counts.

Dense vectors can be much better at capturing synonyms. Car and automobile are synonyms. They have differences in many dimensions. But if you can map the feature vector for car onto 1,000 features and the feature vector for automobile onto 1,000 features, then it may be that the vectors are very similar.

A word with car as a neighbor and a word with automobile as a neighbor, could be similar if they are mapped onto these feature vectors. And the important reason why things like word2vec took off is that in lots of competitions or practical applications they work better.

So short, dense vectors, the main reason they use-- there are lots of theoretical reasons I've just given you which make them better. But in practice, what really counts is that Google and others have found that they actually work better. They give you better-- users like the results better. That's what really counts.

OK, so how can you get these methods for short, dense vectors. Well, the main method that's actually used is these neural or deep learning language models like word2vec or GloVe, which is also available. There are also other methods, like singular value decomposition or latent semantic analysis, which are described in the book in some more detail.

I'm not going to go into these. Because actually these are just not used, or not as widely used as word2vec, or GloVe, or other ones. There are also these things called static embeddings or contextual embeddings, like BERT. We'll look at BERT later on.

For now, I'm not going to cover these. These basically give you a different embedding for any word depending on its context. Whereas what word2vec gives you is, it gives you for a particular word you find 1,000 context it appears in, and it gives you the meaning depending on the use of the 1,000 contexts that it appears in.

OK, we're going to look at word2vec and you can look at GloVe, which is a similar system developed at Stanford University. Word2vec was developed by Google research labs.

And I'm going to recommend that you look at the code page, it's archive, because even now it's recent, but not totally modern. And GloVe is also recent. Have a look at these web pages. You can download the code. And you can download the embeddings, basically a dictionary.

You can look up a word, and for a word like computer, you can get the vector which represents it. Or for cherry, you can get the vector that represents it. And you can yourself compare the vector for cherry and the vector for computer and see how different they are.

So word2vec is a very popular embedding method. It's comparatively fast to train. if you've got a powerful computer, you can actually train it. But you don't actually have to train it, because you can download the results.

You can download the code if you want to do it yourself. Or you can download the outputs, the dictionary that it produces. The way it works, is rather than counting up everything, you have a deep learning model, which tries to predict the best vector for a particular word by seeing how good it is at predicting whether or not a word will actually appear.

So word2vec gives you a number of different parameter settings, a number of different options for learning. And we're going to go one of these particular parameters just by way of example. And this is a skip gram with negative sample or SGNS.

There are other examples. In the textbook there are lots of other examples. I'm just going to quickly go through one example to give you a flavor of how it works. It's only going to be a very brief flavor. And if you need to look--

If you want to develop more embedding examples yourself, then you can do. But I warn you that this is a sort of work which you might want to do for a project. You're certainly not going to be able to do it in one unit on this course.

OK, let's get a flavor of word2vec. It's actually, it was revolutionary at the time, because it came up with a number of ideas which were combined to come up with something which was computationally tractable using the deep learning neural network model.

So the first thing that's different is instead of just counting up how many times a word appears next apricot, the task becomes to train a classifier on a binary prediction task.

So you build a classifier which simply says is the word w likely to show up near to apricot? And the answer is either Yes or No. So you're not so much counting up for a word how many times it occurs, but rather saying, is it likely to appear next to apricot or not likely to appear next apricot.

Now, we don't actually care about what the task is. Because the idea is you come up with a classifier, and the classifier weights are basically a matrix. And the weights are what you use as the word embedding.

You see the classifier results, the weights in the classifier-- we say, represent the meaning of apricot, and these-- so if you build, let's say, 1,000 different classifiers, then those 1,000 classifier weights are your meanings of the word apricot.

Another idea in machine learning, typically you have to have labeled data to do supervised learning. A way of sort of cheating is to say a word C occurring next apricot in the corpus is the label. It's the gold correct answer for supervised learning.

So you don't really need human to do the labeling. Because if humans wrote the text, then when they wrote the text all the words were to appear around the word apricot are the labels of apricot. So sweet or pie are labels for apricot.

And they were sort of human generated because a human did actually write it, but it's self-supervised. It's semi supervised or self-supervised. And the label pie doesn't have to be added by a human afterwards. It's already there in the text.

There's no need for a separate human to be paid to label the data. And this was something that Bengio, and Colobert, and some other researchers worked out independently, that you can take the words in the text as being essentially the labels for the target word.

OK, so we want to predict if a candidate word is a neighbor to the target word. So we treat the target word, t, and a neighboring context word, c, as a positive example.

So if a target word is cherry, and the context word pie appears in it, then we have a positive example of it cherry and pie appear together. And then we randomly sample the other words in the lexicon to get negative examples.

Let's say, information, so cherry and information is a negative example because we've looked up-- we've found a word which is not in the neighborhood of cherry from the dictionary and so it's negative example.

And then we can just use a classifier given positive examples like cherry and pie, and negative examples like cherry and information, and build a classifier. For example, you can use logistic regression to train a classifier to distinguish between those two examples.

And then the weights in the learning model are the embeddings, and that's it. That's all there is to it. So in Skip-Gram, for example, assuming a window of just one or two words on either side, then in a sentence like, "lemon, a tablespoon of apricot jam, a pinch of salt, and so on."

Then around apricot, assuming that the only words around it are a tablespoon of, and jam comma, the target is apricot. The window around it is "tablespoon of apricot jam, a." The only words are c1 is tablespoon, c2 is of, jam is c3, and a c4.

The goal is to train a classifier given a candidate word and context pair like apricot and jam, or apricot and aardvark, assigned a probability. Aardvark is just a random word from the dictionary we've chosen as a negative example. And jam is a positive example.

We assign each pair a probability. And we want to have a positive probability for apricot and jam and a negative probability for apricot and aardvark. Notice that the negative probability is just 1 minus the positive probability, in effect.

We want to build a probabilistic classifier given a test target word, W, like cherry, and its context window of L words. The estimates the probability that word, W, occurs in this window based on similarity of w embeddings to the C embeddings.

OK, so to compute this, we just need the embeddings for all the words. And you can start off with by random numbers as the embeddings. And then we know we gradually improve the embeddings so they're more likely to predict correctly, and less likely to predict wrong.

So we have a set of embeddings, W for all the target words, for aardvark, apricot, and so on down to zebra. That's the vocabulary. And we also have to have the same set of embeddings for all the vocabulary words, so aardvark, apricot, and zebra being in the context.

So apricot is likely to be in the context of apricot. Apricot is likely to be in the context of pie. But apricot is not like to be in the context of aardvark, let's say. And that basically is the model for word2vec. I've glossed over this very quickly. If you want to find out more, then read the textbook.

How do we actually learn these embeddings? Remembering we're using Skip-Gram. There's another algorithm as well. But let's just Skip-Gram. So we have apricot as the target, and tablespoon of jam is the context.

It gives us for the target apricot a number of positive examples, apricot tablespoon, apricot of, apricot jam, apricot a. Notice that some of these positive examples are actually sort of noisy because they contain words like of and a, which are really function words.

So we know that apricot appears next of a lot, and apricot appears next to "a" a lot. But doesn't mean that apricot and of mean the same thing. It's just that they happen to co-occur. But actually we don't have to--

In some sense, you might say you want to get rid of and a because these are stop words. But you don't have to. It comes out in the wash anyway. So these are still plausible positive examples.

We also want some negative examples. And what we do is just get out from the vocabulary some words. And we weight the words with high frequency as well, so that more frequent words are more likely to appear in negative examples.

So for example, we have apricot aardvark, apricot my, apricot where, apricot coaxial. And we tend to have more of the function words like my, and where, and if.

And Furthermore, because we can generate these, because there's lots of vocabulary, we can have more negative examples than positive examples. Let's just say we ought to have twice as many negative examples as positive examples.

And we have an initial set of embedding vectors. And the goal of learning is to adjust the word vectors such that we maximise the similarity of a target word, content word pairs, and minimise the similarity of the negative pairs.

So back in data science you might have come across this. This is a standard way of-- if you've done the deep learning model-- sorry, the deep learning module, then you may start to recognise this.

For those of you who haven't done the deep learning module, don't worry. I'm not going to go into details of exactly how this works. But essentially, you can start off with embeddings which are random numbers.

And then you gradually change these values to maximise to increase the values, which gives you the correct values for target content for positive values and downgrade the ones which are minimal.

And this is done by an algorithm called stochastic gradient descent, which you'll come across in the deep learning module. I'm not going to describe it here, because you don't need to know how it works really.

In terms of text analytics, you can take a black box model and just use it. In terms of a deep learning module, you do have to understand how it works, but I'm not going to do this. We just adjust the word weights to make the positive pairs more likely and negative pairs less likely over the entire training set.

So you have just one example. Apricot jam should co-occur together. So we move the apricot jam matrix up. We increase it. We move. And whereas apricot and matrix are not going to occur, so you should decrease the apricot matrix value.

Or apricot and Tolstoy aren't going to co-occur, so you should downgrade the apricot Tolstoy part of the matrix. So we end up with embeddings for the target matrix. And we end up with embeddings for the context matrix.

So we had to actually end up with two matrices, the target matrix embeddings and the context matrix and readings. Since we only want 1 matrix, it's common just to add these together.

So they represent the meaning of the word is the target embeddings added to the context embeddings. So in summary, how to learn word of it embeddings, you start off with random D dimensional vectors as the initial embeddings.

We start off with random numbers, and then using this, we train a classifier based on similarity. And we have a corpus, and we take the pairs of word that co-occur as positive examples. And then we randomly generate words which don't co-occur as negative examples.

We train the classifier to distinguish this by slowly adjusting all the embeddings to improve the classifier performance. That's basically how neural networks work. You add a very small value to positive examples and subtract a very small value to negative examples.

And do this over and over again until eventually the neural network will correctly predict the positive examples and correctly predict the negative examples. You throw away all this classification. You're not really interested in the classifier at all, but the embeddings are the values in the vectors at the end.

OK, so that's, in a nutshell, how you learn the embeddings. I've said in a nutshell, because really for most text analytics purposes you just want to use embeddings. You don't really care how they were developed.

But it gives you a flavor of how word2vec does it. Because word2vec is a really clever way of coming up with dense vectors, only 1,000 numbers to represent the meanings rather than 50,000 numbers.

The intuitive way of doing it, the understandable way of doing it, using TFIDF or pointwise mutual information, I can understand how that works, but it gives us very large, very sparse vectors. And they're not really practical for computation.

Word2vec, I don't really understand how it works, but it does work. It gives us small, dense vectors. And these are much more computable, they're much easier, much more efficient for computation, and they give us better results.

OK finally, I want to talk a bit about properties of the embeddings. So the kind of neighbors we want depend on the window size. Remember, we looked at just the word before or two words before. And that gives us words which are syntactically similar or similar in syntactic structure.

Back in 1986, the computers at the time, I could only deal with very small windows. I could only look at the word before or after immediately, and that gave me information like of, and in, and to, and for are similar in meaning because they're all prepositions. But that's about all I could compute at that time.

Now, we can look at much larger windows. And we can look at nearest words are related to words in the same semantic field. If you can look at 5 words on either side, and with big computers nowadays you can do that, and it gives us much better meaning similarity.

So if you look at small windows, you'll find that Hogwarts and Sunnydale from Buffy the Vampire Slayer, and Blandings from Jeeves books, these are all similar because they're all names of schools.

Whereas, if you look at larger windows and look at the text of Harry Potter as your corpus, you'll find Hogwarts is related to Dumbledore, and half-blood, and Malfoy. I apologise to people who haven't looked at Harry Potter.

If you're a Harry Potter fan, by the way, come to Leeds. Because quite close-ish to Leeds, maybe 150 miles away from Leeds, which is close in geographical terms, you can go to the Harry Potter experience, and see how the Harry Potter films were made, and go to Hogwarts, and go to Diagon Alley, and things like that. Anyway, that's just an aside.

Another thing you want to be able to do is analogical reasoning. This is a logic puzzle. An apple is to tree as grape is to what? Well, you'd like to be able to see that apple is to tree as grape is to vine. And in the n dimensional space, if we have the apple to tree vector and stick it on to grape, then that should take us to vine.

Or similarly-- and if you look at Michelov paper, or other papers similar to that, we'll see that you can actually take the vector for king, subtract the vector for man, add on the vector for woman, and you get a point in space, which is very similar to the vector for queen.

Or if you take the vector for Paris, subtract the vector for France, add on the vector for Italy, and you get a vector, which is similar to the vector of Rome. In other words, you'll find that the capital of Paris-- sorry, the capital of France is Paris. The capital of Italy is Rome. And that works.

Or if you read the GloVE paper, this isn't the Michelov paper, but there's Stanford University similar system, we see in vector space that the distance between brother and sister is similar to the distance between uncle and aunt, or the distance between man and woman, or the distance between Sir and Madam. They're all related in the same sort of way.

This only really works for frequent words, and for small distances, and certain relations. So it works for countries and capitals. It doesn't work for everything else. So it's a caveat or a constraint on this.

And understanding this is an open area of research. So I'm going to ask you to look at a paper on what we know about this. And some of this, we don't really understand.

You can also use it for historical semantics. For example, if you have Google Books has books from a very large period of time.

If you look at books from the 1900s, and compare it to a sub-corpus of books for 1950s, and compare that to another books sub-corpus of books from the 1990s, we say, for example, that gay was to do with daunting, or flaunting, or daft in the 1900s.

In The 1950s it's to do with bright, and witty, and frolicsome. Whereas, at the end of a century gay appears to be related to bisexual, or homosexual, or lesbian. So gay has changed in meaning over time. And you can see this by comparing three sub corpora from different times.

Embeddings also can reflect cultural bias. We saw, for example, Paris is to France, as Tokyo is x, and x is close to Japan. If you do this from other things, for example, if you ask, father is to doctor, and mother is to x, well it will come back with nurse.

So in n dimensional space, if you take doctor, subtract father, add on mother, you'll get a vector which is very similar to nurse. And that's because in the training data, in the very large set of texts, very often doctor is a male or a father, and very often a nurse is a woman or a mother. And that's a cultural bias.

Another example is, if you take the computer programmer, very often in the training corpus is male or man. What is the equivalent for a woman? It's the homemaker. So if you ask man is to computer programmer as woman is to x? Woman is not a computer programmer.

In the training corpus, it's very often a homemaker. Because the training corpus contains lots of text, from many samples, from many sources and it tends to be gender biased in its source. So algorithms that use embeddings can often be gender biased.

So for example, if you use it for hiring or finding possible candidates for computer programmer, then it will tend to be biased towards men as opposed to women. So you do have to be careful about the meanings in embeddings reflect whatever bias there is in the training data.

And if you take very large corpora of English texts from the web from previous times, then it will be biased. And you will tend to get gender bias, or ethnic bias, or even competent bias. Or dehumanising adjectives, words like barbaric, or monstrous, or bizarre were biased towards Asians in the 1930s.

And if you take the Google Books corpus, which contains books from 1900 up to 2000 or even more recent, you'll see the Google Books corpus is biased. Now, why would you use the Google Books corpus?

Remember, for training these things, you have to have thousands of millions of words of text. And the only very, very large corpora tend to be things like Google Books over time, or possibly the world wide web over time.

There aren't that many corpora which are only very modern. So therefore a lot of these corpora that are used do include these ethical bias issues. OK, I think I'll stop there.

So in summary, we've looked at word meanings. And in machine learning or for computer vision, it may be OK to say the meaning of dog or the meaning of a picture is D-O-G as opposed to C-A-T. But that doesn't really help for the meanings of words.

We want to have vectors of word meanings. And vectors have lots of nice properties that you can measure similarity of two meanings. But for the meaning of wedding can be close to the meaning of marriage because the vectors are similar in terms of cosine metrics.

Taking just raw frequencies isn't ideal, and you may want to normalise these frequencies by using TFIDF or possibly pointwise mutual information. These are two different ways of normalising. And this gives you a vector where many of the values are 0.

And this can be nice because zeros can help in computation. If you multiply out with zeros, then you just get rid of them altogether. The trouble is you tend to get very large vectors, which take a lot of computation.

So ideally, you'd want to have a smaller vector of only 1,000 points as opposed to 50,000 points. If you have only 1,000 features, that makes the computation more straightforward.

Word2vec or GloVe gives you a way of learning embeddings using deep learning, which it gives you very dense and very small vectors. And these vectors are very good in all sorts of ways, particularly in the two words or two documents which means similar things are close together in vector space.

You do have to watch out though that any biases in the training data will be reflected in the embeddings. In particular, gender biases can come up, because if you take a large corpus, and the large corpus includes documents from the past, like the book corpus from Google, that will include books from long ago. And even fairly recent past tends to have gender bias in them.

OK, I'm going to stop there. Thank you, and goodbye for now.

[END]